

ActionScript dictionary

[Overview](#)

[Symbolic operators](#)

[Actions](#)

[Functions](#)

[Objects](#)

[Various](#)

Symbolic operators	
--	-- (decrement)
++	++ (increment)
!	! (logical NOT)
!=	!= (inequality)
%	% (modulo)
%=	%= (modulo assignment)
&	& (bitwise AND)
&&	&& (short-circuit AND)
&=	&= (bitwise AND assignment)
()	() (parentheses)
-	- (minus)
*	* (multiplication)
*=	*= (multiplication assignment)
,	, (comma)
.	. (dot operator)
? :	?: (conditional)
/	/ (division)
//	// (comment delimiter)
/*	/* (comment delimiter)
/=	/= (division assignment)
[]	[] (array access operator)
^	^ (bitwise XOR)
^=	^= (bitwise XOR assignment)
{ }	{} (object initializer)
	 (bitwise OR)
	! (logical NOT)
=	 = (bitwise OR assignment)
~	~ (bitwise NOT)

+	+ (addition)
+=	+= (addition assignment)
<	< (less than)
<<	<< (bitwise left shift)
<<=	<<= (bitwise left shift and assignment)
<=	<= (less than or equal to)
<>	<> (inequality)
=	= (assignment)
-=	-= (negation assignment)
==	== (equality)
>	> (greater than)
>=	>= (greater than or equal to)
>>	>> (bitwise right shift)
>>=	>>= (bitwise right shift and assignment)
>>>	>>> (bitwise unsigned right shift)
>>>=	>>>= (bitwise right shift and assignment)

Actions	
attachMovie	MovieClip.attachMovie
attachSound	Sound.attachSound
break	break
call	call
continue	continue
delete	delete
do...while	do... while
duplicateMovieClip	duplicateMovieClip
else	else
for	for
for.. in	for..in
fscommand	fscommand
function	function
getURL	getURL
gotoAndPlay	gotoAndPlay
gotoAndStop	gotoAndStop
if	if
ifFrameLoaded	ifFrameLoaded

#include	#include
loadMovie	loadMovie
loadVariables	loadVariables
nextFrame	nextFrame
nextScene	nextScene
On	on(mouseEvent)
onClipEvent	onClipEvent
play	play
prevFrame	prevFrame
prevScene	prevScene
print	print
printAsBitmap	printAsBitmap
return	return
set	set
setProperty	setProperty
stop	Sound.stop
stopAllSounds	stopAllSounds
tellTarget	tellTarget
toggleHighQuality	toggleHighQuality
trace	trace
var	var
while	while
with	with

Functions	
Boolean	Boolean (function)
chr	chr
escape (function)	escape
eval	eval
evaluate	evaluate
getProperty	getProperty
getTimer	getTimer
getVersion	getVersion
int	int
isFinite	isFinite
isNaN	isNaN

maxscroll	maxscroll
mbchr	mbchr
mblength	mblength
mbord	mbord
mbsubstring	mbsubstring
newline	newline
Number	Number (function)
ord	ord
parseFloat	parseFloat
parseInt	parseInt
random	random
scroll	scroll
String	String (function)
targetPath	targetPath
unescape	unescape
updateAfterEvent	updateAfterEvent

Objects	
abs	Math.abs
acos	Math.acos
appendChild	XML.appendChild
Array	Array (object)
asin	Math.asin
atan	Math.atan
atan2	Math.atan2
attributes	XML.attributes
BACKSPACE	Key.BACKSPACE
Boolean	Boolean (object)
CAPSLOCK	Key.CAPSLOCK
ceil	Math.ceil
charAt	String.charAt
charCodeAt	String.charCodeAtAt
childNodes	XML.childNodes
cloneNode	XML.cloneNode
close	XMLSocket.close
Color	Color (object)

concat	Array.concat , String.concat
connect	XMLSocket.connect
constructor	Array , Boolean , Color , Date , Number , Object , Sound , String , XML , XMLSocket
CONTROL	Key.CONTROL
cos	Math.cos
createElement	XML.createElement
createTextNode	XML.createTextNode
Date	Date (object)
DELETEKEY	Key.DELETEKEY
docTypeDecl	XML.docTypeDecl
DOWN	Key.DOWN
duplicateMovieClip	MovieClip.duplicateMovieClip
E	Math.E
END	Key.END
ENTER	Key.ENTER
ESCAPE (constant)	Key.ESCAPE
exp	Math.exp
firstChild	XML.firstChild
floor	Math.floor
fromCharCode	String.fromCharCode
getAscii	Key.getAscii
getBeginIndex	Selection.getBeginIndex
getBounds	MovieClip.getBounds
getBytesLoaded	MovieClip.getBytesLoaded
getBytesTotal	MovieClip.getBytesTotal
getCaretIndex	Selection.getCaretIndex
getCode	Key.getCode
getDate	Date.getDate
getDay	Date.getDay
getEndIndex	Selection.getEndIndex
getFocus	Selection.getFocus
getFullYear	Date.getFullYear
getHours	Date.getHours
getMilliseconds	Date.getMilliseconds
getMinutes	Date.getMinutes
getMonth	Date.getMonth
getPan	Sound.getPan

getRGB	Color.setRGB
getSeconds	Date.getSeconds
getTime	Date.getTime
getTimezoneOffset	Date.getTimezoneOffset
getTransform	Color.getTransform , Sound.getTransform
getURL	MovieClip.getURL
getUTCDate	Date.getUTCDate
getUTCDay	Date.getUTCDay
getUTCFullYear	Date.getUTCFullYear
getUTCHours	Date.getUTCHours
getUTCMilliseconds	Date.getUTCMilliseconds
getUTCMinutes	Date.getUTCMinutes
getUTCMonth	Date.getUTCMonth
getUTCSeconds	Date.getUTCSeconds
getVolume	Sound.getVolume
getYear	Date.getYear
globalToLocal	MovieClip.globalToLocal
gotoAndPlay	MovieClip.gotoAndPlay
gotoAndStop	MovieClip.gotoAndStop
hasChildNodes	XML.haschildNodes
hide	Mouse.hide
hitTest	MovieClip.hitTest
HOME	Key.HOME
indexOf	String.indexOf
INSERT	Key.INSERT
insertBefore	XML.insertBefore
isDown	Key.isDown
isToggled	Key.isToggled
join	Array.join
Key	Key (object)
lastChild	XML.lastChild
lastIndexOf	String.indexOf
LEFT	Key.LEFT
length	length , Array.length , String.length
LN2	Math.LN2
LN10	Math.LN10
load	XML.load

loaded	XML.loaded
loadVariables	MovieClip.loadVariables
localToGlobal	MovieClip.localToGlobal
log	Math.log
LOG2E	Math.LOG2E
LOG10E	Math.LOG10E
Math	Math (object)
max	Math.max
MAX_VALUE	Number.MAX_VALUE
min	Math.min
MIN_VALUE	Number.MIN_VALUE
Mouse	Mouse (object)
MovieClip	MovieClip (object)
NaN	NaN , Number.NaN
NEGATIVE_INFINITY	Number.NEGATIVE_INFINITY
nextFrame	MovieClip.nextFrame
nextSibling	XML.nextSibling
nodeName	XML.nodeName
nodeType	XML.nodeType
nodeValue	XML.nodeValue
Number	Number (object)
Object	Object (object)
onClose	XMLSocket.onClose
onConnect	XMLSocket.onConnect
OnLoad	XML.onLoad
onXML	XMLSocket.onXML
parentNode	XML.parentNode
parseXML	XML.parseXML
PGDN	Key.PGDN
PGUP	Key.PGUP
PI	Math.PI
play	MovieClip.play
pop	Array.pop
POSITIVE_INFINITY	Number.POSITIVE_INFINITY
pow	Math.pow
prevFrame	MovieClip.prevFrame
previousSibling	XML.previousSibling

push	Array.push
random	Math.random
removeMovieClip	removeMovieClip , MovieClip.removeMovieClip
removeNode	XML.removeNode
reverse	Array.reverse
RIGHT	Key.RIGHT
round	Math.round
Selection	Selection (object)
send	XML.send , XMLSocket.send
sendAndLoad	XML.sendAndLoad
setDate	Date.setDate
setFocus	Selection.setFocus
setFullYear	Date.setFullYear
setHours	Date.setHours
setMilliseconds	Date.setMilliseconds
setMinutes	Date.setMinutes
setMonth	Date.setMonth
setPan	Sound.setPan
setRGB	Color.setRGB
setSeconds	Date.setSeconds
setSelection	Selection.setSelection
setTime	Date.setTime
setTransform	Color.setTransform , Sound.setTransform
setUTCDate	Date.setUTCDate
setUTCFullYear	Date.setUTCFullYear
setUTCHours	Date.setUTCHours
setUTCMilliseconds	Date.setUTCMilliseconds
setUTCMinutes	Date.setUTCMinutes
setUTCMonth	Date.setUTCMonth
setUTCSeconds	Date.setUTCSeconds
setVolume	Sound.setVolume
setYear	Date.setYear
shift (method)	Array.shift
SHIFT (constant)	Key.SHIFT
show	Mouse.show
sin	Math.sin
slice	Array.slice , String.slice

sort	Array.sort
Sound	Sound (object)
SPACE	Key.SPACE
splice	Array.splice
split	String.split
sqrt	Math.sqrt
SQRT1_2	Math.SQRT1_2
SQRT2	Math.SQRT2
start	Sound.start
startDrag	startDrag , MovieClip.startDrag
status	XML.status
stop	MovieClip.stop
stopDrag	stopDrag , MovieClip.stopDrag
String	String (object)
String	" " (string delimiter)
substr	String.substr
substring	substring , String.substring
swapDepths	MovieClip.swapDepths
TAB	Key.TAB
tan	Math.tan
toLowerCase	String.toLowerCase
toString	Array.toString , Boolean.toString , Date.toString , Number.toString , Object.toString , XML.toString
toUpperCase	String.toUpperCase
unloadMovie	unloadMovie , MovieClip.unloadMovie
unshift	Array.shift
UP	Key.UP
UTC	Date.UTC
valueOf	Boolean.valueOf , Number.valueOf , Object.valueOf
XML	XML (object)
xmlDecl	XML.xmlDecl
XMLSocket	XMLSocket (object)

Various

add	add
and	and
_alpha	_alpha

<code>_currentframe</code>	<code><u>_currentframe</u></code>
<code>_droptarget</code>	<code><u>_droptarget</u></code>
<code>eq</code>	<code>eq (equal—string specific)</code>
<code>_focusrect</code>	<code><u>_focusrect</u></code>
<code>_framesloaded</code>	<code><u>_framesloaded</u></code>
<code>ge</code>	<code>ge (greater than or equal to—string specific)</code>
<code>gt</code>	<code>gt (greater than —string specific)</code>
<code>_height</code>	<code><u>_height</u></code>
<code>_highquality</code>	<code><u>_highquality</u></code>
<code>Infinity</code>	<code><u>Infinity</u></code>
<code>le</code>	<code>le (less than or equal to — string specific)</code>
<code>lt</code>	<code>le (less than or equal to — string specific)</code>
<code>_name</code>	<code><u>_name</u></code>
<code>ne</code>	<code>ne (not equal — string specific)</code>
<code>new (operator)</code>	<code><u>new</u></code>
<code>not</code>	<code><u>not</u></code>
<code>null</code>	<code><u>null</u></code>
<code>or (logical OR)</code>	<code><u>or</u></code>
<code>_parent</code>	<code><u>_parent</u></code>
<code>_quality</code>	<code><u>_quality</u></code>
<code>_root</code>	<code><u>_root</u></code>
<code>_rotation</code>	<code><u>_rotation</u></code>
<code>_soundbuftime</code>	<code><u>_soundbuftime</u></code>
<code>_target</code>	<code><u>_target</u></code>
<code>this</code>	<code><u>this</u></code>
<code>_totalframes</code>	<code><u>_totalframes</u></code>
<code>typeof</code>	<code><u>typeof</u></code>
<code>_url</code>	<code><u>_url</u></code>
<code>_visible</code>	<code><u>_visible</u></code>
<code>void</code>	<code><u>void</u></code>
<code>_width</code>	<code><u>_width</u></code>
<code>_x</code>	<code><u>_x</u></code>
<code>_xmouse</code>	<code><u>_xmouse</u></code>
<code>_xscale</code>	<code><u>_xscale</u></code>
<code>_y</code>	<code><u>_y</u></code>
<code>_ymouse</code>	<code><u>_ymouse</u></code>
<code>_yscale</code>	<code><u>_yscale</u></code>

SEARCH



ActionScript dictionary: Overview

This portion of the ActionScript Reference Guide describes the syntax and use of ActionScript elements in Flash 5 and later versions. To use examples in a script, copy the example text from ActionScript Dictionary Help and paste it in the Actions panel in Expert Mode.

The dictionary lists all ActionScript elements—operators, keywords, statements, actions, properties, functions, objects, and methods. For an overview of all dictionary entries, see [Contents of the dictionary](#); the tables in this section are a good starting point for looking up symbolic operators or methods whose object class you don't know.

ActionScript follows the ECMA-262 standard (the specification written by the European Computer Manufacturers Association) unless otherwise noted.

There are two types of entries in this dictionary:

- Individual entries for operators, keywords, functions, variables, properties, methods, and statements
- Object entries, which provide general detail about predefined objects

Use the information in the sample entries to interpret the structure and conventions used in these two types of entries.

[Sample entry for most ActionScript elements](#)

[Sample entry for objects](#)

[Contents of the dictionary](#)

Contents of the dictionary

All dictionary entries are listed alphabetically. However, some operators are symbols, and are presented in ASCII order. In addition, methods that are associated with an object are listed along with the object's name—for example, the `abs` method of the `Math` object is listed as `Math.abs`.

The following two tables will help you locate these elements. The first table lists the symbolic operators in the order in which they occur in the dictionary. The second table lists all other ActionScript elements.

Note: For precedence and associativity of operators, see Appendix A.

Symbolic operators

--	-- (decrement)
++	++ (increment)
!	! (logical NOT)
!=	!= (inequality)
%	% (modulo)
%=	%= (modulo assignment)
&	& (bitwise AND)
&&	&& (short-circuit AND)
&=	&= (bitwise AND assignment)
()	() (parentheses)
-	- (minus)
*	* (multiplication)
*=	*= (multiplication assignment)
,	, (comma)
.	. (dot operator)
?:	?: (conditional)
/	/ (division)
//	// (comment delimiter)
/*	/* (comment delimiter)
/=	/= (division assignment)
[]	[] (array access operator)
^	^ (bitwise XOR)
^=	^= (bitwise XOR assignment)
{ }	{} (object initializer)
	 (bitwise OR)
	 (logical NOT)
=	 = (bitwise OR assignment)
~	~ (bitwise NOT)

+	+ (addition)
+=	+= (addition assignment)
<	< (less than)
<<	<< (bitwise left shift)
<<=	<<= (bitwise left shift and assignment)
<=	<= (less than or equal to)
<>	<> (inequality)
=	= (assignment)
-=	-= (negation assignment)
==	== (equality)
>	> (greater than)
>=	>= (greater than or equal to)
>>	>> (bitwise right shift)
>>=	>>= (bitwise right shift and assignment)
>>>	>>> (bitwise unsigned right shift)
>>>=	>>>= (bitwise right shift and assignment)

The following table lists all ActionScript elements that are not symbolic operators.

ActionScript element	See entry
abs	Math.abs
acos	Math.acos
add	add
and	and
_alpha	_alpha
appendChild	XML.appendChild
Array	Array (object)
asin	Math.asin
atan	Math.atan
atan2	Math.atan2
attachMovie	MovieClip.attachMovie
attachSound	Sound.attachSound
attributes	XML.attributes
BACKSPACE	Key.BACKSPACE
Boolean	Boolean (function) , Boolean (object)
break	break
call	call

CAPSLOCK	Key.CAPSLOCK
ceil	Math.ceil
charAt	String.charAt
charCodeAt	String.charCodeAtAt
childNodes	XML.childNodes
chr	chr
cloneNode	XML.cloneNode
close	XMLSocket.close
Color	Color (object)
concat	Array.concat , String.concat
connect	XMLSocket.connect
constructor	Array , Boolean , Color , Date , Number , Object , Sound , String , XML , XMLSocket
continue	continue
CONTROL	Key.CONTROL
cos	Math.cos
createElement	XML.createElement
createTextNode	XML.createTextNode
_currentframe	_currentframe
Date	Date (object)
delete	delete
DELETEKEY	Key.DELETEKEY
docTypeDecl	XML.docTypeDecl
do...while	do... while
DOWN	Key.DOWN
_droptarget	_droptarget
duplicateMovieClip	duplicateMovieClip , MovieClip.duplicateMovieClip
E	Math.E
else	else
END	Key.END
ENTER	Key.ENTER
eq	eq (equal—string specific)
escape (function)	escape
ESCAPE (constant)	Key.ESCAPE
eval	eval
evaluate	evaluate
exp	Math.exp

firstChild	XML.firstChild
floor	Math.floor
_focusrect	_focusrect
for	for
for.. in	for..in
_framesloaded	_framesloaded
fromCharCode	String.fromCharCode
fscommand	fscommand
function	function
ge	ge (greater than or equal to—string specific)
getAscii	Key.getAscii
getBeginIndex	Selection.getBeginIndex
getBounds	MovieClip.getBounds
getBytesLoaded	MovieClip.getBytesLoaded
getBytesTotal	MovieClip.getBytesTotal
getCaretIndex	Selection.getCaretIndex
getCode	Key.getCode
getDate	Date.getDate
getDay	Date.getDay
getEndIndex	Selection.getEndIndex
getFocus	Selection.getFocus
getFullYear	Date.getFullYear
getHours	Date.getHours
getMilliseconds	Date.getMilliseconds
getMinutes	Date.getMinutes
getMonth	Date.getMonth
getPan	Sound.getPan
getProperty	getProperty
getRGB	Color.setRGB
getSeconds	Date.getSeconds
getTime	Date.getTime
getTimer	getTimer
getTimezoneOffset	Date.getTimezoneOffset
getTransform	Color.getTransform , Sound.getTransform
getURL	getURL , MovieClip.getURL
getUTCDate	Date.getUTCDate

getUTCDay	Date.getUTCDay
getUTCFullYear	Date.getUTCFullYear
getUTCHours	Date.getUTCHours
getUTCMilliseconds	Date.getUTCMilliseconds
getUTCMinutes	Date.getUTCMinutes
getUTCMonth	Date.getUTCMonth
getUTCSeconds	Date.getUTCSeconds
getVersion	getVersion
getVolume	Sound.getVolume
getYear	Date.getYear
globalToLocal	MovieClip.globalToLocal
gotoAndPlay	gotoAndPlay , MovieClip.gotoAndPlay
gotoAndStop	gotoAndStop , MovieClip.gotoAndStop
gt	gt (greater than —string specific)
hasChildNodes	XML.hasChildNodes
_height	_height
hide	Mouse.hide
_highquality	_highquality
hitTest	MovieClip.hitTest
HOME	Key.HOME
if	if
ifFrameLoaded	ifFrameLoaded
#include	#include
indexOf	String.indexOf
Infinity	Infinity
INSERT	Key.INSERT
insertBefore	XML.insertBefore
int	int
isDown	Key.isDown
isFinite	isFinite
isNaN	isNaN
isToggled	Key.isToggled
join	Array.join
Key	Key (object)
lastChild	XML.lastChild
lastIndexOf	String.indexOf

le	le (less than or equal to — string specific)
LEFT	Key.LEFT
length	length , Array.length , String.length
LN2	Math.LN2
LN10	Math.LN10
load	XML.load
loaded	XML.loaded
loadMovie	loadMovie , MovieClip.loadMovie
loadVariables	loadVariables , MovieClip.loadVariables
localToGlobal	MovieClip.localToGlobal
log	Math.log
LOG2E	Math.LOG2E
LOG10E	Math.LOG10E
lt	le (less than or equal to — string specific)
Math	Math (object)
max	Math.max
maxscroll	maxscroll
MAX_VALUE	Number.MAX_VALUE
mbchr	mbchr
mblength	mblength
mbord	mbord
mbsubstring	mbsubstring
min	Math.min
MIN_VALUE	Number.MIN_VALUE
Mouse	Mouse (object)
MovieClip	MovieClip (object)
_name	_name
NaN	NaN , Number.NaN
ne	ne (not equal — string specific)
NEGATIVE_INFINITY	Number.NEGATIVE_INFINITY
new (operator)	new
newline	newline
nextFrame	nextFrame , MovieClip.nextFrame
nextScene	nextScene
nextSibling	XML.nextSibling
nodeName	XML.nodeName

nodeType	XML.nodeType
nodeValue	XML.nodeValue
not	not
null	null
Number	Number (function) , Number (object)
Object	Object (object)
On	on(mouseEvent)
onClipEvent	onClipEvent
onClose	XMLSocket.onClose
onConnect	XMLSocket.onConnect
OnLoad	XML.onLoad
onXML	XMLSocket.onXML
or (logical OR)	or
ord	ord
_parent	_parent
parentNode	XML.parentNode
parseFloat	parseFloat
parseInt	parseInt
parseXML	XML.parseXML
PGDN	Key.PGDN
PGUP	Key.PGUP
PI	Math.PI
play	play , MovieClip.play
pop	Array.pop
POSITIVE_INFINITY	Number.POSITIVE_INFINITY
pow	Math.pow
prevFrame	prevFrame , MovieClip.prevFrame
previousSibling	XML.previousSibling
prevScene	prevScene
print	print
printAsBitmap	printAsBitmap
push	Array.push
_quality	_quality
random	random , Math.random
removeMovieClip	removeMovieClip , MovieClip.removeMovieClip
removeNode	XML.removeNode

return	return
reverse	Array.reverse
RIGHT	Key.RIGHT
_root	_root
_rotation	_rotation
round	Math.round
scroll	scroll
Selection	Selection (object)
send	XML.send , XMLSocket.send
sendAndLoad	XML.sendAndLoad
set	set
setDate	Date.setDate
setFocus	Selection.setFocus
setFullYear	Date.setFullYear
setHours	Date.setHours
setMilliseconds	Date.setMilliseconds
setMinutes	Date.setMinutes
setMonth	Date.setMonth
setPan	Sound.setPan
setProperty	setProperty
setRGB	Color.setRGB
setSeconds	Date.setSeconds
setSelection	Selection.setSelection
setTime	Date.setTime
setTransform	Color.setTransform , Sound.setTransform
setUTCDate	Date.setUTCDate
setUTCFullYear	Date.setUTCFullYear
setUTCHours	Date.setUTCHours
setUTCMilliseconds	Date.setUTCMilliseconds
setUTCMinutes	Date.setUTCMinutes
setUTCMonth	Date.setUTCMonth
setUTCSeconds	Date.setUTCSeconds
setVolume	Sound.setVolume
setYear	Date.setYear
shift (method)	Array.shift
SHIFT (constant)	Key.SHIFT

show	Mouse.show
sin	Math.sin
slice	Array.slice , String.slice
sort	Array.sort
Sound	Sound (object)
_soundbuftime	_soundbuftime
SPACE	Key.SPACE
splice	Array.splice
split	String.split
sqrt	Math.sqrt
SQRT1_2	Math.SQRT1_2
SQRT2	Math.SQRT2
start	Sound.start
startDrag	startDrag , MovieClip.startDrag
status	XML.status
stop	stop , MovieClip.stop , Sound.stop
stopAllSounds	stopAllSounds
stopDrag	stopDrag , MovieClip.stopDrag
String	String (function) , String (object) , " " (string delimiter)
substr	String.substr
substring	substring , String.substring
swapDepths	MovieClip.swapDepths
TAB	Key.TAB
tan	Math.tan
_target	_target
targetPath	targetPath
tellTarget	tellTarget
this	this
toggleHighQuality	toggleHighQuality
toLowerCase	String.toLowerCase
toString	Array.toString , Boolean.toString , Date.toString , Number.toString , Object.toString , XML.toString
_totalframes	_totalframes
toUpperCase	String.toUpperCase
trace	trace
typeof	typeof

unescape	unescape
unloadMovie	unloadMovie , MovieClip.unloadMovie
unshift	Array.shift
UP	Key.UP
updateAfterEvent	updateAfterEvent
_url	_url
UTC	Date.UTC
valueOf	Boolean.valueOf , Number.valueOf , Object.valueOf
var	var
_visible	_visible
void	void
while	while
_width	_width
with	with
_x	_x
XML	XML (object)
xmlDecl	XML.xmlDecl
XMLSocket	XMLSocket (object)
_xmouse	_xmouse
_xscale	_xscale
_y	_y
_ymouse	_ymouse
_yscale	_yscale

-- (decrement)

Syntax

--expression
expression--

Arguments `expression` A variable, number, element in an array, or property of an object.

Description Operator; a pre-decrement and post-decrement unary operator that subtracts 1 from the `expression`. The pre-decrement form of the operator (`--expression`) subtracts 1 from `expression` and returns the result. The post-decrement form of the operator (`expression--`) subtracts 1 from the `expression` and returns the initial value of the `expression` (the result prior to the subtraction).

Player Flash 4 or later.

Example The pre-decrement form of the operator decrements `x` to 2 (`x - 1 = 2`), and returns the result as `y`:

```
x = 3;
```

```
y = --x
```

The post-decrement form of the operator decrements `x` to 2 (`x - 1 = 2`), and returns the original value (`x = 3`) as the result `y`:

```
If x = 3;
```

```
y = x--
```

[CONTENTS](#) 

ActionScript dictionary

[Overview](#)[Symbolic operators](#)[Actions](#)[Functions](#)[Objects](#)[Various](#)

Symbolic operators

--	-- (decrement)
----	--------------------------------

++	++ (increment)
----	--------------------------------

!	! (logical NOT)
---	---------------------------------

!=	!= (inequality)
----	---------------------------------

%	% (modulo)
---	----------------------------

%=	%= (modulo assignment)
----	--

&	& (bitwise AND)
---	-------------------------------------

&&	&& (short-circuit AND)
----	--

&=	&= (bitwise AND assignment)
----	---

()	() (parentheses)
----	----------------------------------

-	- (minus)
---	---------------------------

*	* (multiplication)
---	------------------------------------

*=	*= (multiplication assignment)
----	--

,	, (comma)
---	---------------------------

.	. (dot operator)
---	----------------------------------

?:	?: (conditional)
----	----------------------------------

/	/ (division)
---	------------------------------

//	// (comment delimiter)
----	--

/*	/* (comment delimiter)
----	--

/=	/= (division assignment)
----	--

[]	[] (array access operator)
----	--

^	^ (bitwise XOR)
---	---------------------------------

^=	^= (bitwise XOR assignment)
----	---

{ }	{} (object initializer)
-----	---

	 (bitwise OR)
--	--------------------------------

	! (logical NOT)
--	---------------------------------

=	 = (bitwise OR assignment)
---	--

~	~ (bitwise NOT)
+	+ (addition)
+=	+= (addition assignment)
<	< (less than)
<<	<< (bitwise left shift)
<<=	<<= (bitwise left shift and assignment)
<=	<= (less than or equal to)
<>	<> (inequality)
=	= (assignment)
-=	-= (negation assignment)
==	== (equality)
>	> (greater than)
>=	>= (greater than or equal to)
>>	>> (bitwise right shift)
>>=	>>= (bitwise right shift and assignment)
>>>	>>> (bitwise unsigned right shift)
>>>=	>>>= (bitwise right shift and assignment)

Actions	
attachMovie	MovieClip.attachMovie
attachSound	Sound.attachSound
break	break
call	call
continue	continue
delete	delete
do...while	do... while
duplicateMovieClip	duplicateMovieClip
else	else
for	for
for.. in	for..in
fscommand	fscommand
function	function
getURL	getURL
gotoAndPlay	gotoAndPlay
gotoAndStop	gotoAndStop

if	<u>if</u>
ifFrameLoaded	<u>ifFrameLoaded</u>
#include	<u>#include</u>
loadMovie	<u>loadMovie</u>
loadVariables	<u>loadVariables</u>
nextFrame	<u>nextFrame</u>
nextScene	<u>nextScene</u>
On	<u>on(mouseEvent)</u>
onClipEvent	<u>onClipEvent</u>
play	<u>play</u>
prevFrame	<u>prevFrame</u>
prevScene	<u>prevScene</u>
print	<u>print</u>
printAsBitmap	<u>printAsBitmap</u>
return	<u>return</u>
set	<u>set</u>
setProperty	<u>setProperty</u>
stop	<u>Sound.stop</u>
stopAllSounds	<u>stopAllSounds</u>
tellTarget	<u>tellTarget</u>
toggleHighQuality	<u>toggleHighQuality</u>
trace	<u>trace</u>
var	<u>var</u>
while	<u>while</u>
with	<u>with</u>

Functions	
Boolean	<u>Boolean (function)</u>
chr	<u>chr</u>
escape (function)	<u>escape</u>
eval	<u>eval</u>
evaluate	<u>evaluate</u>
getProperty	<u>getProperty</u>
getTimer	<u>getTimer</u>
getVersion	<u>getVersion</u>

int	int
isFinite	isFinite
isNaN	isNaN
maxscroll	maxscroll
mbchr	mbchr
mblength	mblength
mbord	mbord
mbsubstring	mbsubstring
newline	newline
Number	Number (function)
ord	ord
parseFloat	parseFloat
parseInt	parseInt
random	random
scroll	scroll
String	String (function)
targetPath	targetPath
unescape	unescape
updateAfterEvent	updateAfterEvent

Objects

abs	Math.abs
acos	Math.acos
appendChild	XML.appendChild
Array	Array (object)
asin	Math.asin
atan	Math.atan
atan2	Math.atan2
attributes	XML.attributes
BACKSPACE	Key.BACKSPACE
Boolean	Boolean (object)
CAPSLOCK	Key.CAPSLOCK
ceil	Math.ceil
charAt	String.charAt
charCodeAt	String.charCodeAt

childNodes	XML.childNodes
cloneNode	XML.cloneNode
close	XMLSocket.close
Color	Color (object)
concat	Array.concat , String.concat
connect	XMLSocket.connect
constructor	Array , Boolean , Color , Date , Number , Object , Sound , String , XML , XMLSocket
CONTROL	Key.CONTROL
cos	Math.cos
createElement	XML.createElement
createTextNode	XML.createTextNode
Date	Date (object)
DELETEKEY	Key.DELETEKEY
docTypeDecl	XML.docTypeDecl
DOWN	Key.DOWN
duplicateMovieClip	MovieClip.duplicateMovieClip
E	Math.E
END	Key.END
ENTER	Key.ENTER
ESCAPE (constant)	Key.ESCAPE
exp	Math.exp
firstChild	XML.firstChild
floor	Math.floor
fromCharCode	String.fromCharCode
getAscii	Key.getAscii
getBeginIndex	Selection.getBeginIndex
getBounds	MovieClip.getBounds
getBytesLoaded	MovieClip.getBytesLoaded
getBytesTotal	MovieClip.getBytesTotal
getCaretIndex	Selection.getCaretIndex
getCode	Key.getCode
getDate	Date.getDate
getDay	Date.getDay
getEndIndex	Selection.getEndIndex
getFocus	Selection.getFocus
getFullYear	Date.getFullYear

getHours	Date.getHours
getMilliseconds	Date.getMilliseconds
getMinutes	Date.getMinutes
getMonth	Date.getMonth
getPan	Sound.getPan
getRGB	Color.setRGB
getSeconds	Date.getSeconds
getTime	Date.getTime
getTimezoneOffset	Date.getTimezoneOffset
getTransform	Color.getTransform , Sound.getTransform
getURL	MovieClip.getURL
getUTCDate	Date.getUTCDate
getUTCDay	Date.getUTCDay
getUTCFullYear	Date.getUTCFullYear
getUTCHours	Date.getUTCHours
getUTCMilliseconds	Date.getUTCMilliseconds
getUTCMinutes	Date.getUTCMinutes
getUTCMonth	Date.getUTCMonth
getUTCSeconds	Date.getUTCSeconds
getVolume	Sound.getVolume
getYear	Date.getYear
globalToLocal	MovieClip.globalToLocal
gotoAndPlay	MovieClip.gotoAndPlay
gotoAndStop	MovieClip.gotoAndStop
hasChildNodes	XML.hasChildNodes
hide	Mouse.hide
hitTest	MovieClip.hitTest
HOME	Key.HOME
indexOf	String.indexOf
INSERT	Key.INSERT
insertBefore	XML.insertBefore
isDown	Key.isDown
isToggled	Key.isToggled
join	Array.join
Key	Key (object)
lastChild	XML.lastChild

lastIndexOf	String.indexOf
LEFT	Key.LEFT
length	length , Array.length , String.length
LN2	Math.LN2
LN10	Math.LN10
load	XML.load
loaded	XML.loaded
loadVariables	MovieClip.loadVariables
localToGlobal	MovieClip.localToGlobal
log	Math.log
LOG2E	Math.LOG2E
LOG10E	Math.LOG10E
Math	Math (object)
max	Math.max
MAX_VALUE	Number.MAX_VALUE
min	Math.min
MIN_VALUE	Number.MIN_VALUE
Mouse	Mouse (object)
MovieClip	MovieClip (object)
NaN	NaN , Number.NaN
NEGATIVE_INFINITY	Number.NEGATIVE_INFINITY
nextFrame	MovieClip.nextFrame
nextSibling	XML.nextSibling
nodeName	XML.nodeName
nodeType	XML.nodeType
nodeValue	XML.nodeValue
Number	Number (object)
Object	Object (object)
onClose	XMLSocket.onClose
onConnect	XMLSocket.onConnect
OnLoad	XML.onLoad
onXML	XMLSocket.onXML
parentNode	XML.parentNode
parseXML	XML.parseXML
PGDN	Key.PGDN
PGUP	Key.PGUP

PI	Math.PI
play	MovieClip.play
pop	Array.pop
POSITIVE_INFINITY	Number.POSITIVE_INFINITY
pow	Math.pow
prevFrame	MovieClip.prevFrame
previousSibling	XML.previousSibling
push	Array.push
random	Math.random
removeMovieClip	removeMovieClip , MovieClip.removeMovieClip
removeNode	XML.removeNode
reverse	Array.reverse
RIGHT	Key.RIGHT
round	Math.round
Selection	Selection (object)
send	XML.send , XMLSocket.send
sendAndLoad	XML.sendAndLoad
setDate	Date.setDate
setFocus	Selection.setFocus
setFullYear	Date.setFullYear
setHours	Date.setHours
setMilliseconds	Date.setMilliseconds
setMinutes	Date.setMinutes
setMonth	Date.setMonth
setPan	Sound.setPan
setRGB	Color.setRGB
setSeconds	Date.setSeconds
setSelection	Selection.setSelection
setTime	Date.setTime
setTransform	Color.setTransform , Sound.setTransform
setUTCDate	Date.setUTCDate
setUTCFullYear	Date.setUTCFullYear
setUTCHours	Date.setUTCHours
setUTCMilliseconds	Date.setUTCMilliseconds
setUTCMinutes	Date.setUTCMinutes
setUTCMonth	Date.setUTCMonth

setUTCSeconds	Date.setUTCSeconds
setVolume	Sound.setVolume
setYear	Date.setYear
shift (method)	Array.shift
SHIFT (constant)	Key.SHIFT
show	Mouse.show
sin	Math.sin
slice	Array.slice , String.slice
sort	Array.sort
Sound	Sound (object)
SPACE	Key.SPACE
splice	Array.splice
split	String.split
sqrt	Math.sqrt
SQRT1_2	Math.SQRT1_2
SQRT2	Math.SQRT2
start	Sound.start
startDrag	startDrag , MovieClip.startDrag
status	XML.status
stop	MovieClip.stop
stopDrag	stopDrag , MovieClip.stopDrag
String	String (object)
String	" " (string delimiter)
substr	String.substr
substring	substring , String.substring
swapDepths	MovieClip.swapDepths
TAB	Key.TAB
tan	Math.tan
toLowerCase	String.toLowerCase
toString	Array.toString , Boolean.toString , Date.toString , Number.toString , Object.toString , XML.toString
toUpperCase	String.toUpperCase
unloadMovie	unloadMovie , MovieClip.unloadMovie
unshift	Array.unshift
UP	Key.UP
UTC	Date.UTC

valueOf	Boolean.valueOf , Number.valueOf , Object.valueOf
XML	XML (object)
xmlDecl	XML.xmlDecl
XMLSocket	XMLSocket (object)

Various	
add	add
and	and
_alpha	_alpha
_currentframe	_currentframe
_droptarget	_droptarget
eq	eq (equal—string specific)
_focusrect	_focusrect
_framesloaded	_framesloaded
ge	ge (greater than or equal to—string specific)
gt	gt (greater than —string specific)
_height	_height
_highquality	_highquality
Infinity	Infinity
le	le (less than or equal to — string specific)
lt	lt (less than or equal to — string specific)
_name	_name
ne	ne (not equal — string specific)
new (operator)	new
not	not
null	null
or (logical OR)	or
_parent	_parent
_quality	_quality
_root	_root
_rotation	_rotation
_soundbuftime	_soundbuftime
_target	_target
this	this
_totalframes	_totalframes

typeof	typeof
_url	_url
_visible	_visible
void	void
_width	_width
_x	_x
_xmouse	_xmouse
_xscale	_xscale
_y	_y
_ymouse	_ymouse
_yscale	_yscale

++ (increment)

Syntax

```
++expression  
expression++
```

Arguments `expression` A variable, number, element in an array, or property of an object.

Description Operator; a pre-increment and post-increment unary operator that adds 1 to the `expression`. The pre-increment form of the operator (`++expression`) adds 1 to the `expression` and returns the result. The post-increment form of the operator (`expression++`) adds 1 to the `expression` and returns the initial value of the `expression` (the result prior to the addition).

The pre-increment form of the operator increments `x` to 2 (`x + 1 = 2`), and returns the result as `y`:

```
x = 1;  
y = ++x
```

The post-increment form of the operator increments `x` to 2 (`x + 1 = 2`), and returns the original value (`x = 1`) as the result `y`:

```
x = 1;  
y = x++;
```

Player Flash 4 or later.

Example The following example uses `++` as a pre-increment operator with a while statement.

```
i = 0  
while(i++ < 5){  
    // this section will execute five times  
}
```

The following example uses `++` as a pre-increment operator:

```
var a = [];  
var i = 0;  
while (i < 10) {  
    a.push(++i);  
}  
trace(a.join());
```

This script prints the following:

1,2,3,4,5,6,7,8,9

The following example uses `++` as a post-increment operator:

```
var a = [];  
var i = 0;  
while (i < 10) {  
    a.push(i++);  
}  
trace(a.join());
```

This script prints the following:

0,1,2,3,4,5,6,7,8,9

©1995-2001 Macromedia, Inc. [All rights reserved.](#)

[Privacy policy](#) | [Contact us](#) | [Feedback](#)

! (logical NOT)

Syntax

`!expression`

Arguments `expression` A variable or evaluated expression.

Description Operator (logical); inverts the Boolean value of a variable or expression. If `expression` is a variable with an absolute or converted value `true`, `!variable` the value of `! expression` is `false`. If the expression `x && y` evaluates to `false`, the expression `!(x && y)` evaluates to `true`. This operator is identical to the `not` operator that was used in Flash 4.

Player Flash 4 or later.

Example In the following example the variable `happy` is set to `false`, the `if` condition evaluates the condition `!happy`, and if the condition is `true`, `trace` sends a string to the Output window.

```
happy = false; if (!happy){ trace("don't worry be happy"); }
```

The following illustrates the results of the `!` operator:

`! true` returns `false`

`! false` returns `true`

CONTENTS 

!= (inequality)

Syntax

`expression1 != expression2`

Arguments `expression1`, `expression2` Numbers, strings, Booleans, variables, objects, arrays, or functions.

Description Operator (equality); tests for the exact opposite of the `==` operator. If `expression1` is equal to `expression2`, the result is false. As with the `==` operator, the definition of equal depends on the data types being compared.

- ▶ Numbers, strings, and Boolean values are compared by value.
- ▶ Variables, objects, arrays, and functions are compared by reference.

Player Flash 5 or later.

Example The following example illustrates the results of the `!=` operator.

`5 != 8` returns true

`5 != 5` returns false

The following example illustrates the use of the `!=` operator in an if statement:

```
a = "David"; b = "Fool" if (a != b){ trace("David is not a fool"); }
```

See also

[== \(equality\)](#)

[CONTENTS](#) 

% (modulo)

Syntax

`expression1 % expression2`

Arguments `expression1`, `expression2` Numbers, integers, floating-point numbers, or strings that convert to a numeric value.

Description Operator (arithmetic); calculates the remainder of `expression1` divided by `expression2`. If either of the `expression` arguments are nonnumeric, the modulo operator attempts to convert them to numbers.

Player Flash 4 or later. In Flash 4 files, the % operator is expanded in the SWF file as `x - int(x/y) * y`, and may not be as fast or as accurate as the Flash 5 Player implementation.

Example The following is a numeric example of using the % operator:

12 % 5 returns 2

4.3 % 2.1 returns 0.1

[CONTENTS](#) 

%= (modulo assignment)

Syntax

`expression1 %= expression2`

Arguments `expression1,expression2` Integers and variables.

Description Operator (assignment); assigns `expression1` the value of `expression1 % expression2`.

Player Flash 4 or later.

Example The following illustrates using the %= operator with variables and numbers:

`x %= y` is the same as `x = x % y`

If `x = 14` and `y = 5` then

`x %= 5` returns 4

See also [% \(modulo\)](#)

[CONTENTS](#) 

& (bitwise AND)

Syntax

```
expression1 & expression2
```

Arguments expression1, expression2 Any number.

Description Operator (bitwise); converts expression1 and expression2 to 32-bit unsigned integers, and performs a Boolean AND operation on each bit of the integer arguments. The result is a new 32-bit unsigned integer.

Player Flash 5 or later. In Flash 4 the & operator was used for concatenating strings. In Flash 5 the & operator is a bitwise AND, and the add and + operators concatenate strings. Flash 4 files that use the & operator are automatically updated to use add when brought into the Flash 5 authoring environment.

[CONTENTS](#) 

&& (short-circuit AND)

Syntax

`expression1 && expression2`

Arguments `expression1`, `expression2` Numbers, strings, variables, or functions.

Description Operator (logical); performs a Boolean operation on the values of one or both of the expressions. Causes the Flash interpreter to evaluate `expression1` (the left expression) and returns `false` if the expression evaluates to `false`. If `expression1` evaluates to `true`, `expression2` (the right) is evaluated. If `expression2` evaluates to `true`, the final result is `true`; otherwise, it is `false`.

Player Flash 4 or later.

Example This example assigns the values of the evaluated expressions to the variables `winner` and `loser` in order to perform a test:

```
winner = (chocolateEggs >=10) && (jellyBeans >=25);
loser = (chocolateEggs <=1) && (jellyBeans <= 5);
if (winner) {
    alert = "You Win the Hunt!";
    if (loser) {
        alert = "Now THAT'S Unhappy Hunting!";
    }
} else {
    alert = "We're all winners!";
}
```

[CONTENTS](#) 

&= (bitwise AND assignment)

Syntax

```
expression1 &= expression2
```

Arguments expression1, expression2 Integers and variables.

Description Operator (bitwise assignment); assigns expression1 the value of expression1 & expression2.

Player Flash 5 or later.

Example The following illustrates using the &= operator with variables and numbers:

x &= y is the same as x = x & y

If x = 15 and y = 9 then

x &= 9 returns 9

See also [& \(bitwise AND\)](#)

[CONTENTS](#) 

() (parentheses)

Syntax

```
(expression1, expression2);  
function(functionCall1, ..., functionCallN);
```

Arguments expression1, expression2 Numbers, strings, variables, or text.

function The function to be performed on the contents of the parentheses.

functionCall1...functionCallN A series of functions to execute before the result is passed to the function outside the parentheses.

Description Operator (general); performs a grouping operation on one or more arguments, or surrounds one or more arguments and passes the results a parameter to a function outside the parentheses.

Usage 1: Performs a grouping operation on one or more expressions to control the order of execution of the operators in the expression. This operator overrides the automatic precedence order, and causes the expressions within the parentheses to be evaluated first. When parentheses are nested, Flash evaluates the contents of the innermost parentheses before the contents of the outer ones.

Usage 2: Surrounds one or more arguments and passes them as parameters to the function outside the parentheses.

Player Flash 4 or later.

Example (Usage 1) The following statements illustrate the use of parentheses to control the order of execution of expressions. (The result appears below each statement.)

```
( 2 + 3 ) * ( 4 + 5 ) 45  
2 + ( 3 * ( 4 + 5 ) ) 29  
2 + ( 3 * 4 ) + 5 19
```

(Usage 2) The following example illustrates the use of parentheses with a function:

```
getDate();  
invoice(item, amount);
```

See also

[with](#)

CONTENTS 

- (minus)

Syntax (Negation) `-expression`

(Subtraction) `expression1 - expression2`

Arguments `expression1`, `expression2` Any number.

Description Operator (arithmetic); used for negating or subtracting. When used for negating, it reverses the sign of the numerical `expression`. When used for subtracting, it performs an arithmetic subtraction on two numerical expressions, subtracting `expression2` from `expression1`. When both expressions are integers, the difference is an integer. When either or both expressions are floating-point numbers, the difference is a floating-point number.

Player Flash 4 or later.

Example (Negation) This statement reverses the sign of the expression `2 + 3`:

```
- ( 2 + 3 )
```

The result is -5.

(Subtraction) This statement subtracts the integer 2 from the integer 5:

```
5 - 2
```

The result is 3, which is an integer.

(Subtraction) : This statement subtracts the floating-point number 1.5 from the floating-point number 3.25:

```
put 3.25 - 1.5
```

The result is 1.75, which is a floating-point number.

[CONTENTS](#) 

* (multiplication)

Syntax

`expression1 * expression2`

Arguments `expression1`, `expression2` Integers or floating-point numbers.

Description Operator (arithmetic); multiplies two numerical expressions. If both expressions are integers, the product is an integer. If either or both expressions are floating-point numbers, the product is a floating-point number.

Player Flash 4 or later.

Example This statement multiplies the integers 2 and 3:

```
2 * 3
```

The result is 6, which is an integer.

Example This statement multiplies the floating-point numbers 2.0 and 3.1416:

```
2.0 * 3.1416
```

The result is 6.2832, which is a floating-point number.

[CONTENTS](#) 

***= (multiplication assignment)**

Syntax

`expression1 *= expression2`

Arguments `expression1`, `expression2` Integers, floating-point numbers, or strings.

Description Operator (assignment); assigns `expression1` the value of `expression1 * expression2`.

Player Flash 4 or later.

Example The following illustrates using the *= operator with variables and numbers:

`x *= y` is the same as `x = x * y`

If `x = 5` and `y = 10` then

`x *= 10` returns 50

See also [* \(multiplication\)](#)

[CONTENTS](#) 

, (comma)

Syntax

`expression1, expression2`

Arguments `expression` Any number, variable, string, array element, or other data.

Description Operator; instructs Flash to evaluate `expression1`, then `expression2`, and return the value of `expression2`. This operator is primarily used with the `for` loop statement.

Player Flash 4 or later.

Example The following code sample uses the comma operator:

```
var a=1, b=2, c=3;
```

This is equivalent to writing the following:

```
var a=1; var b=2; var c=3;
```

[CONTENTS](#) 

. (dot operator)

Syntax

```
object.property_or_method  
instancename.variable  
instancename.childinstance.variable
```

Arguments **object** An instance of an object. Some objects require that instances be created using the constructor for that object. The object can be any of the predefined ActionScript objects or a custom object. This argument is always to the left of the dot (.) operator.

property_or_method The name of a property or method associated with an object. All of the valid method and properties for the predefined objects are listed in the Method and Property summary tables for that object. This argument is always to the right of the dot (.) operator.

instancename The name of a movie clip instance.

childinstance An movie clip instance that is a child of the main movie clip.

variable A variable in a movie clip.

Description Operator; used to navigate movie clip hierarchies in order to access nested child movie clips, variables, or properties. The dot operator is also used to test or set the properties of an object, execute a method of an object, or create a data structure.

Player Flash 4 or later.

See also

[\[\] \(array access operator\)](#)

Example This statement identifies the current value of the variable `hairColor` by the movie clip `person`:

```
person.hairColor
```

This is equivalent to the following Flash 4 syntax:

```
/person:hairColor
```

Example The following code illustrates how the dot operator can be used to create a structure of an array:

```
account.name = "Gary Smith";  
account.address = "123 Main St ";  
account.city = "Any Town";  
account.state = "CA";  
account.zip = "12345";
```

[CONTENTS](#) 

?: (conditional)

Syntax `expression1 ? expression2 : expression3`

Arguments `expression1` An expression that evaluates to a Boolean value, usually a comparison expression.

`expression2`, `expression3` Values of any type.

Description Operator (conditional); instructs Flash to evaluate `expression1`, and return the value of `expression2` if `expression1` is `true`; otherwise return the value of the `expression3`.

Player Flash 4 or later.

[CONTENTS](#) 

/ (division)

Syntax

`expression1 / expression2`

Arguments `expression` Any number.

Description Operator (arithmetic); divides `expression1` by `expression2`. The expression arguments and results of the division operation are treated and expressed as double-precision floating-point numbers.

Player Flash 4 or later.

Example This statement divides the floating-point number 22.0 by 7.0 and then displays the result in the Output window:

```
trace(22.0 / 7.0);
```

The result is 3.1429, which is a floating-point number.

[CONTENTS](#) 

// (comment delimiter)

Syntax

// comment

Arguments comment Text that is not part of the code, and should be ignored by the interpreter.

Description Comment; indicates the beginning of a script comment. Any text that appears between the comment delimiter // and the end-of-line character is interpreted as a comment and ignored by the ActionScript interpreter.

Player Flash 1 or later.

Example

[CONTENTS](#) 

/* (comment delimiter)

Syntax

```
/* comment */  
/*  
* comment  
* comment  
*/
```

Arguments `comment` Any text

Description Comment; indicates one or more lines of script comments. Any text that appears between the opening comment tag `/*` and the closing comment tag `*/`, is interpreted as a comment and ignored by the ActionScript interpreter. Use the first syntax to identify single-line comments, and use the second syntax to identify comments on multiple successive lines. Leaving off the closing tag `*/` when using this form of comment delimiter causes the ActionScript compiler to return an error message.

Player Flash 5 or later.

See also [// \(comment delimiter\)](#)

CONTENTS 

/= (division assignment)

Syntax

`expression1 /= expression2`

Arguments `expression1,expression2` Integers, floating-point numbers, or strings.

Description Operator (assignment); assigns `expression1` the value of `expression1 / expression2`.

Player Flash 4 or later.

Example The following illustrates using the /= operator with variables and numbers:

```
x /= y is the same as x = x /y
x = 10; y = 2; x /= y; // x now contains the value 5
```

[CONTENTS](#) 

[] (array access operator)

Syntax

```
myArray["a0", "a1", ... "aN"];  
object[value1, value2, ...valueN];
```

Arguments myArray The name of an array.

a0, a1,...aN Elements in an array.

value1, 2,...N Names of properties.

Description Operator; creates a new object initializing the properties specified in the arguments, or initializes new array with the elements (a0) specified in the arguments.

The created object has the generic Object object as its prototype. Using this operator is the same as calling new Object and populating the properties using the assignment operator. Using this operator is an alternative to using the new operator, which allows for the quick and convenient creation of objects.

Player Flash 4 or later.

Example The following example code samples are two different ways of creating a new empty Array object:

```
myArray = [ ] ; myArray = new Array();
```

The following is an example of a simple array:

```
myArray = [ "red", "orange", "yellow", "green", "blue", "purple" ]  
myArray[0]="red" myArray[1]="yellow" myArray[2]="green"  
myArray[3]="blue" myArray[4]="purple"
```

[CONTENTS](#) 

^(bitwise XOR)

Syntax

`expression1 ^ expression2`

Arguments `expression1,expression2` Any number.

Description Operator (bitwise); converts `expression1` and `expression2` to 32-bit unsigned integers, and returns a 1 in each bit position where the corresponding bits in `expression1` or `expression1`, but not both, are 1.

Player Flash 5 or later.

Example

`15 ^ 9` returns 6
(`1111 ^ 1001 = 0110`)

[CONTENTS](#) 

^= (bitwise XOR assignment)

Syntax

`expression1 ^= expression2`

Arguments `expression1,expression2` Integers and variables.

Description Operator (compound assignment); assigns `expression1` the value of `expression1 ^ expression2`.

Player Flash 5 or later.

Example The following is an example of a ^= operation:

```
// 15 decimal = 1111 binary
x = 15; // 9 decimal = 1001 binary
x ^= y; // returns x ^ y (0110 binary)
```

The following illustrates using the ^= operator with variables and numbers:

`x ^= y` is the same as `x = x ^ y` If `x = 15` and `y = 9` then `15 ^= 9` returns `6`

See also [^\(bitwise XOR\)](#)

[CONTENTS](#) 

{ } (object initializer)

Syntax

```
object {name1: value1,  
       name1: value2,  
       ...  
       nameN: valueN };
```

Arguments `object` The object to create.

`name1,2,...N` The name of the property.

`value1,2,...N` The corresponding value for each `name` property.

Description Operator; creates a new object and initializes it with the specified `name` and `value` property pairs. The created object has the generic `Object` object as its prototype. Using this operator is the same as calling `new Object` and populating the property pairs using the assignment operator. Using this operator is an alternative to using the `new` operator, which allows for the quick and convenient creation of objects.

Player Flash 5 or later.

Example The following code shows how an empty object can be created using the object initializer operator and using the `new Object`:

```
object = {} ; object = new Object();
```

The following creates an object `account` initializing the properties `name`, `address`, `city`, `state`, `zip`, and `balance`:

```
account = { name: "John Smith", address: "123 Main Street", city:  
"Blossomville", state: "California", zip: "12345", balance: "1000"  
};
```

The following example shows how array and object initializers can be nested within each other:

```
person = { name: "Peter Piper", children: [ "Jack", "Jill", "Moe", ]  
};
```

The following example is another way of using the information in the previous example above, with the same results:

```
person = new Person(); person.name = 'John Smith'; person.children =  
new Array(); person.children[0] = 'Jack'; person.children[1] =  
'Jill'; person.children[2] = 'Moe';
```

See also

[\[\] \(array access operator\)](#)

[new Object \(object\)](#)

[CONTENTS](#) 

| (bitwise OR)

Syntax

`expression1 | expression2`

Arguments `expression1,expression2` Any number.

Description Operator (bitwise); converts `expression1` and `expression2` to 32-bit unsigned integers, and returns a 1 in each bit position where the corresponding bits of either `expression1` or `expression2` are 1.

Player Flash 5 or later.

Example The following is an example of aFlash - ActionScript Dictionary : | (bitwise OR)| (bitwise OR)| (bitwise OR)| (bitwise OR) operation. Note that 15 is 1111 binary:

```
// 15 decimal = 1111 binary
x = 15;
// 9 decimal = 1001 binary
y = 9;
// x | y = binary
z = x | y;
z = 15
```

The following is another way of expressing the preceding example:

```
15 | 9 returns 15
(1111 | 0011 = 1111)
```

[CONTENTS](#) 

|= (bitwise OR assignment)

Syntax

expression1 |= expression2

Arguments expression1,expression2 Integers and variables.

Description Operator (assignment); assigns expression1 the value of expression1 | expression2.

Player Flash 5 or later.

Example The following illustrates using the |= operator with variables and numbers:

x |= y is the same as x = x | y

If x = 15 and y = 9 then

x |= 9 returns 15

See also [| \(bitwise OR\)](#)

[CONTENTS](#) 

~ (bitwise NOT)

Syntax

~ expression

Arguments expression Any number.

Description Operator (bitwise); converts the expression to a 32-bit unsigned integer, then inverts the bits. Or, simply said, changes the sign of a number and subtracts 1.

A bitwise NOT operation changes the sign of a number and subtracts 1.

Player Flash 5 or later.

Example The following is a numerical explanation of a bitwise NOT operation performed on a variable:

~a, returns -1 if a = 0, and returns -2 if a = 1, thus:

~0=-1 and ~1=-2

[CONTENTS](#) 

+ (addition)

Syntax

`expression1 + expression2`

Arguments `expression1,expression2` Integers, numbers, floating-point numbers, or strings.

Description Operator; adds numeric expressions or concatenates strings. If one expression is a string, all other expressions are converted to strings and concatenated.

If both expressions are integers, the sum is an integer; if either or both expressions are floating-point numbers, the sum is a floating-point number.

Player Flash 4; Flash 5 or later. In Flash 5, + is a numeric operator or string concatenator depending on the data type of the argument. In Flash 4, + is only a numeric operator. Flash 4 files brought into the Flash 5 authoring environment undergo a conversion process to maintain data type integrity. The first example below illustrates the conversion process.

Example The following illustrates the conversion of a Flash 4 file containing a numeric quality comparison:

Flash 4 file:

`x + y`

Converted Flash 5 file:

`Number(x) + Number(y)`

This statement adds the integers 2 and 3 and then displays the resulting integer, 5, in the Output window:

`trace (2 + 3);`

This statement adds the floating-point numbers 2.5 and 3.25 and displays the result, 5.7500, a floating-point number, in the Output window:

`trace (2.5 + 3.25);`

This statement concatenates two strings and displays the result, "today is my birthday," in the Output window:

`"today is my" + "birthday"`

See also

[add](#)

[CONTENTS](#) 

+= (addition assignment)

Syntax

```
expression1 += expression2
```

Arguments expression1,expression2 Integers, floating-point numbers, or strings.

Description Operator (compound assignment); assigns expression1 the value of expression1 + expression2. This operator also performs string concatenation.

Player Flash 4 or later.

Example This following illustrates a numeric use of the += operator:

x += y is the same as x = x + y

If x = 5 and y = 10 then

x += 10 returns 15

This example illustrates using the += operator with a string expression:

```
x = "My name is" x += "Mary"
```

The result for the above code is as follows:

```
"My name is Mary"
```

See also

[+ \(addition\)](#)

[CONTENTS](#) 

< (less than)

Syntax

```
expression1 < expression2
```

Arguments expression1,expression2 Numbers or strings.

Description Operator (comparison); compares two expressions and determines whether expression1 is less than expression2 (true), or whether expression1 is greater than or equal to expression2 (false). String expressions are evaluated and compared based on the number of characters in the string.

Player Flash 4; Flash 5 or later. In Flash 5 < is a comparison operator capable of handling various data types. In Flash 4 < is an numeric operator. Flash 4 files brought into the Flash 5 authoring environment undergo a conversion process to maintain data type integrity. The first example below illustrates the conversion process.

Example The following illustrates the conversion of a Flash 4 file containing a numeric quality comparison.

Flash 4 file:

```
x < y
```

Converted Flash 5 file:

```
Number(x) < Number(y)
```

The following examples illustrate true and false returns for both numbers and strings:

3 < 10 or "Al" < "Jack" return true

10 < 3 or "Jack" < "Al" return false

[CONTENTS](#) 

<< (bitwise left shift)

Syntax `expression1 << expression2`

Arguments `expression1` A number, string, or expression to be shifted left.

`expression2` A number, string, or expression that converts to an integer from 0 to 31.

Description Operator (bitwise); converts `expression1` and `expression2` to 32-bit integers, and shifts all of the bits in `expression1` to the left by the number of places specified by the integer resulting from the conversion of `expression2`. The bit positions that are emptied as a result of this operation are filled in with 0. Shifting a value left by one position is the equivalent of multiplying it by 2.

Player Flash 5 or later.

Example The following example shifts the integer 1 ten bits to the left:

```
x = 1 << 10
```

The result of this operation is `x = 1024`. This is because 1 decimal equals 1 binary, 1 binary shifted left by 10 is 10000000000 binary, and 10000000000 binary is 1024 decimal.

This following example shifts the integer 7 eight bits to the left:

```
x = 7 << 8
```

The result of this operation is `x = 1792`. This is because 7 decimal equals 111 binary, 111 binary shifted left by 8 bits is 11100000000 binary, and 11100000000 binary is 1792 decimal.

See also

[>>= \(bitwise right shift and assignment\)](#)

[CONTENTS](#) 

<<= (bitwise left shift and assignment)

Syntax

expression1 <<= expression2

Arguments expression1 A number, string, or expression to be shifted left.

expression2 A number, string, or expression that converts to an integer from 0 to 31.

Description Operator (compound assignment); this operator performs a bitwise left shift operation and stores the contents as a result in expression1.

Player Flash 5 or later.

Example The following two expressions are equivalent:

A <<= B A = (A << B)

See also

[<< \(bitwise left shift\)](#)

[>>= \(bitwise right shift and assignment\)](#)

CONTENTS 

<= (less than or equal to)

Syntax

```
expression1 <= expression2
```

Arguments expression1,expression2 Number or strings.

Description Operator (comparison); compares two expressions and determines whether expression1 is less than or equal to expression2 (true), or whether expression1 is greater than expression2 (false).

Player Flash 4; Flash 5 or later. In Flash 5 <= is a comparison operator capable of handling various data types. In Flash 4 <= is an numeric operator. Flash 4 files brought into the Flash 5 authoring environment undergo a conversion process to maintain data type integrity. The first example below illustrates the conversion process.

Example The following illustrates the conversion of a Flash 4 file containing a numeric quality comparison.

Flash 4 file:

```
x <= y
```

Converted Flash 5 file:

```
Number(x) <= Number(y)
```

The following examples illustrate true and false results for both numbers and strings:

5 <= 10 or "Al" <= "Jack" returns true

10<= 5 or "Jack" <= "Al" returns false

[CONTENTS](#) 

<> (inequality)

Syntax

```
expression1 <> expression2
```

Arguments expression1,expression2 Numbers, strings, Booleans, variables, objects, arrays, or functions.

Description Operator (equality); tests for the exact opposite of the == operator. If expression1 is equal to expression2, the result is false. As with the == operator, the definition of equal depends on the data types being compared:

- ▶ Numbers, strings, and Boolean values are compared by value.
- ▶ Variables, objects, arrays, and functions are compared by reference.

This operator has been deprecated in Flash 5, and users are encouraged to make use of the new != operator.

Player Flash 2 or later.

See also

[!= \(inequality\)](#)

[CONTENTS](#) 

= (assignment)

Syntax

```
expression1 = expression2
```

Arguments `expression1` A variable, element of an array, or property of an object.

`expression2` A value of any type.

Description Operator (assignment); assigns the type of `expression2` (the argument on the right) to the variable, array element, or property in `expression1`.

Player Flash 4; Flash 5 or later. In Flash 5 = is an assignment operator and the == operator is used to evaluate equality. In Flash 4 = is a numeric equality operator. Flash 4 files brought into the Flash 5 authoring environment undergo a conversion process to maintain data type integrity. The first example below illustrates the conversion process.

Example The following illustrates the conversion of a Flash 4 file containing a numeric quality comparison.

Flash 4 file:

```
x = y
```

Converted Flash 5 file:

```
Number(x) == Number(y)
```

The following example uses the assignment operator to assign the number data type to the variable x:

```
x = 5
```

The following example uses the assignment operator to assign the string data type to the variable x:

```
x = "hello"
```

[CONTENTS](#) 

-= (negation assignment)

Syntax

`expression1 -= expression2`

Arguments `expression1,expression2` Integers, floating-point numbers, or strings.

Description Operator (compound assignment); assigns `expression1` the value of `expression1 - expression2`.

Player Flash 4 or later.

Example The following illustrates using the -= operator with variables and numbers:

`x -= y` is the same as `x = x - y`

If `x = 5` and `y = 10` then

`x -= 10` returns -5

[CONTENTS](#) 

== (equality)

Syntax

```
expression1 == expression2
```

Arguments expression1,expression2 Numbers, strings, Booleans, variables, objects, arrays, or functions.

Description Operator (equality); tests two expressions for equality. The result is true if the expressions are equal.

The definition of equal depends on the data type of the argument:

- ▶ Numbers, strings, and Boolean values are compared by value, and are considered equal if they have the same value. For instance, two strings are equal if they have the same number of characters.
- ▶ Variables, objects, arrays, and functions are compared by reference. Two variables are equal if they refer to the same object, array, or function. Two separate arrays are never considered equal, even if they have the same number of elements.

Player Flash 5 or later.

Example The following example uses the == operator with an if statement:

```
a = "David" , b = "David"; if (a == b){ trace("David is David"); }
```

[CONTENTS](#) 

> (greater than)

Syntax

```
expression1 > expression2
```

Arguments expression1,expression2 Integers, floating-point numbers, or strings.

Description Operator (comparison); compares two expressions and determines whether expression1 is greater than expression2 (true), or whether expression1 is less than or equal to expression2 (false).

Player Flash 4; Flash 5 or later. In Flash 5 > is a comparison operator capable of handling various data types. In Flash 4 > is an numeric operator. Flash 4 files brought into the Flash 5 authoring environment undergo a conversion process to maintain data type integrity. The example below illustrates the conversion process.

Example The following illustrates the conversion of a Flash 4 file containing a numeric quality comparison.

Flash 4 file:

```
x > y
```

Converted Flash 5 file:

```
Number(x) > Number(y)
```

[CONTENTS](#) 

>= (greater than or equal to)

Syntax

```
expression1 >= expression2
```

Arguments expression1, expression2 Strings, integers, or floating-point numbers.

Description Operator (comparison); compares two expressions and determines whether expression1 is greater than or equal to expression2 (true), or whether expression1 is less than expression2 (false).

Player Flash 4; Flash 5 or later. In Flash 5 >= is a comparison operator capable of handling various data types. In Flash 4 >= is a numeric operator. Flash 4 files brought into the Flash 5 authoring environment undergo a conversion process to maintain data type integrity. The example below illustrates the conversion process.

Example The following illustrates the conversion of a Flash 4 file containing a numeric quality comparison.

Flash 4 file:

```
x >= y
```

Converted Flash 5 file:

```
Number(x) >= Number(y)
```

[CONTENTS](#) 

>> (bitwise right shift)

Syntax

`expression1 >> expression2`

Arguments `expression1` A number, string, or expression to be shifted right.

`expression2` A number, string, or expression that converts to an integer from 0 to 31.

Description Operator (bitwise); converts `expression1` and `expression2` to 32-bit integers, and shifts all of the bits in `expression1` to the right by the number of places specified by the integer resulting from the conversion of `expression2`. Bits that are shifted off to the right are discarded. To preserve the sign of the original `expression`, the bits on the left are filled in with 0 if the most significant bit (the bit farthest to the left) of `expression1` is 0, and filled in with 1 if the most significant bit is 1. Shifting a value right by one position is the equivalent of dividing by 2 and discarding the remainder.

Player Flash 5 or later.

Example The following example converts 65535 to a 32-bit integer, and shifts it eight bits to the right:

```
x = 65535 >> 8
```

The result of the above operation is as follows:

```
x = 255
```

This is because 65535 decimal equals 1111111111111111 binary (sixteen 1's), 1111111111111111 binary shifted right by eight bits is 11111111 binary, and 11111111 binary is 255 decimal. The most significant bit is 0 because the integers are 32-bit, so the fill bit is 0.

The following example converts -1 to a 32-bit integer and shifts it one bit to the right:

```
x = -1 >> 1
```

The result of the above operation is as follows:

```
x = -1
```

This is because -1 decimal equals 11111111111111111111111111111111 binary (thirty-two 1's), shifting right by one bit causes the least significant (bit farthest to the right) to be discarded and the most significant bit to be filled in with 1. The result is 11111111111111111111111111111111 (thirty-two 1's) binary, which represents the 32-bit integer -1.

See also

[>>= \(bitwise right shift and assignment\)](#)

[CONTENTS](#) 

>>= (bitwise right shift and assignment)

Syntax

`expression1 >>= expression2`

Arguments `expression1` A number, string, or expression to be shifted left.

`expression2` A number, string, or expression that converts to an integer from 0 to 31.

Description Operator (compound assignment); this operator performs a bitwise right shift operation and stores the contents as a result in `expression1`.

Player Flash 5 or later.

Example The following two expressions are equivalent:

`A >>= B A = (A >> B)`

The following commented code uses the bitwise operator `>>=`. It is also an example of using all bitwise operators.

```
function convertToBinary(number) { var result = ""; for (var i=0; i<32; i++) { // Extract least significant bit using bitwise AND var lsb = number & 1; // Add this bit to our result string result = (lsb ? "1" : "0") + result; // Shift number right by one bit, to see next bit } number >>= 1; return result; } convertToBinary(479)
//Returns the string 000000000000000000000000111011111 //The above string is the binary representation of the decimal number 479.
```

See also

[<< \(bitwise left shift\)](#)

[CONTENTS](#) 

MovieClip.attachMovie

Syntax

```
anyMovieClip.attachMovie(idName, newname, depth);
```

Arguments `idName` The name of the movie in the library to attach. This is the name entered in the Identifier field in the Symbol Linkage Properties dialog box.

`newname` A unique instance name for the movie clip being attached.

`depth` An integer specifying the depth level where the movie is placed.

Description Method; creates a new instance of a movie in the library and attaches it to the movie specified by `anyMovieClip`. Use the `removeMovieClip` or `unloadMovie` action or method to remove a movie attached with `attachMovie`.

Player Flash 5 or later.

See also

[removeMovieClip](#) [unloadMovie](#) [MovieClip.removeMovieClip](#)
[MovieClip.unloadMovie](#)

CONTENTS 

Sound.attachSound

Syntax

```
mySound.attachSound( "idName" );
```

Arguments `idName` The name for the new instance of the sound. This is the same as the name entered for the identifier in the Symbol Linkage Properties dialog box. This argument must be enclosed in " " (quotation marks).

Description Method; attaches the sound specified in the `idName` argument to the specified Sound object. The sound must be in the library of the current movie and specified for export in the Symbol Linkage Properties dialog box. You must call `Sound.start` to start playing the sound.

Player Flash 5 or later.

See also

[Sound.start](#)

[CONTENTS](#) 

break

Syntax

```
break;
```

Arguments

None.

Description Action; appears within a loop (for, for..in, do...while or while). The break action instructs Flash to skip the rest of the loop body, stop the looping action, and execute the statement following the loop statement. Use the break action to break out of a series of nested loops.

Player Flash 4 or later.

Example The following example uses the break action to exit an otherwise infinite loop:

```
i = 0; while (true) { if (i >= 100) { break; } i++; }
```

[CONTENTS](#) 

call

Syntax

```
call(frame);
```

Arguments `frame` The name or number of the frame to call into the context of the script.

Description Action; switches the context from the current script to the script attached to the frame being called. Local variables will not exist once the script is finished executing.

Player Flash 4 or later. This action is deprecated in Flash 5, and it is recommended that you use the function action.

See also

[function](#)

[CONTENTS](#) 

continue

Syntax

```
continue;
```

Arguments None.

Description Action; appears within several types of loop statements.

In a `while` loop, `continue` causes Flash to skip the rest of the loop body and jump to the top of the loop, where the condition is tested.

In a `do...while` loop, `continue` causes Flash to skip the rest of the loop body and jump to the bottom of the loop, where the condition is tested.

In a `for` loop, `continue` causes Flash to skip the rest of the loop body and jump to the evaluation of the `for` loop's post-expression.

In a `for...in` loop, `continue` causes Flash to skip the rest of the loop body and jump back to the top of the loop, where the next value in the enumeration is processed.

Player Flash 4 or later.

See also

[do... while](#)

[for](#)

[for..in](#)

[while](#)

[CONTENTS](#) 

delete

Syntax

```
delete (reference);
```

Arguments reference The name of variable or object to eliminate.

Description Operator; destroys the object or variable specified as the reference, and returns true if the object was successfully deleted; otherwise returns false. This operator is useful for freeing up memory used by scripts, although, delete is an operator, it is typically used as a statement:

```
delete x;
```

The delete operator may fail and return false if the reference does not exist, or may not be deleted. Predefined objects and properties, and variables declared with var, may not be deleted.

Player Flash 5 or later.

Example The following example creates an object, uses it, and then deletes it once it is no longer needed:

```
account = new Object(); account.name = 'Jon'; account.balance =  
10000; ... delete account;
```

The following example deletes a property of an object:

```
// create the new object "account" account = new Object(); // assign  
property name to the account account.name = 'Jon'; // delete the  
property delete account.name;
```

The following is another example of deleting an object property:

```
// create an Array object with length 0  
array = new Array();  
// Array.length is now 1  
array[0] = "abc";  
// add another element to the array, Array.length is now 2  
array[1] = "def";  
// add another element to array, Array.length is now 3  
array[2] = "ghi";  
// array[2] is deleted, but Array.length is not changed,  
delete array[2];
```

The following example illustrates the behavior of delete on object references:

```
// create a new object, and assign the variable ref1 to refer to  
the object ref1 = new Object(); ref1.name = "Jody"; // copy the  
reference variable into a new variable, and delete ref1 ref2 =  
ref1; delete ref1;
```

If ref1 had not been copied into ref2, the object would have been deleted when we deleted ref1, because there would be no references to it. If we were to delete ref2, there would no longer be any references to the object, and it would be destroyed and the memory it was using would be made available.

See also

[var](#)

©1995-2001 Macromedia, Inc. [All rights reserved.](#)

[Privacy policy](#) | [Contact us](#) | [Feedback](#)

do... while

Syntax

```
do {  
    statement;  
} while (condition);
```

Arguments `condition` The condition to evaluate.

`statement` The statement to execute as long as `condition` evaluates to true.

Description Action; executes the statements, and then evaluates the condition in a loop for as long as the condition is true.

Player Flash 4 or later.

See also

[break](#)

[continue](#)

[CONTENTS](#) 

duplicateMovieClip

Syntax

```
duplicateMovieClip(target, newname, depth);
```

Arguments `target` The target path of the movie to duplicate.

`newname` A unique identifier for the duplicate movie clip.

`depth` The depth level of the movie clip. The depth level is the stacking order that determines how movie clips and other objects appear when they overlap. The first movie clip that your create, or instance that you drag onto the Stage, is assigned a depth of level 0. You must assign each successive or duplicated movie clip a different depth level to prevent it from replacing movies on occupied levels or the original movie clip.

Description Action; creates an instance of a movie clip while the movie is playing. Duplicate movie clips always start at frame 1, no matter what frame the original movie clip was on. Variables in the parent movie clip are not copied into the duplicate movie clip. If the parent movie clip is deleted the duplicate movie clip is also deleted. Use the `removeMovieClip` action or method to delete a movie clip instance created with `duplicateMovieClip`.

Player Flash 4 or later.

Example This statement duplicates the movie clip instance `flower` ten times. The variable `i` is used to create a new instance name and a depth.

```
on(release) {
    amount = 10;
    while(amount>0) {
        duplicateMovieClip (_root.flower, "mc" + i, i);
        setProperty("mc" + i, _x, random(275));
        setProperty("mc" + i, _y, random(275));
        setProperty("mc" + i, _alpha, random(275));
        setProperty("mc" + i, _xscale, random(50));
        setProperty("mc" + i, _yscale, random(50));
        i = i + 1;
        amount = amount-1;
    }
}
```

See also

[removeMovieClip](#) [MovieClip.removeMovieClip](#)

CONTENTS 

else

Syntax

```
else {statement(s)};
```

Arguments statement(s) An alternative series of statements to run if the condition specified in the if statement is false.

Description Action; specifies the actions, clauses, arguments, or other conditional to run if the initial if statement returns false.

Player Flash 4 or later.

See also

[if](#)

[CONTENTS](#) 

for

Syntax

```
for(init; condition; next); {  
    statement;  
}
```

Arguments **init** An expression to evaluate before beginning the looping sequence, typically an assignment expression. A var statement is also permitted for this argument.

condition An expression that evaluates to true or false. The condition is evaluated before each loop iteration; the loop exits when the condition evaluates to false.

next An expression to evaluate after each loop iteration; usually an assignment expression using the ++ (increment) or -- (decrement) operators.

statement A statement within the body of the loop to execute.

Description Action; a loop construct that evaluates the **init** (initialize) expression once, and then begins a looping sequence by which, as long as the **condition** evaluates to true, **statement** is executed and the next expression is evaluated.

Some properties can not be enumerated by the **for** or **for..in** actions. For example, the built-in methods of the Array object (Array.sort and Array.reverse) are not included in the enumeration of an Array object, and movie clip properties, such as **_x** and **_y**, are not enumerated.

Player Flash 5 or later.

Example The following example uses **for** to add the elements in an array:

```
for(i=0; i<10; i++) { array [i] = (i + 5)*10; }
```

Returns the following array:

```
[50, 60, 70, 80, 90, 100, 110, 120, 130, 140]
```

The following is an example of using **for** to perform the same action repeatedly. In the code below, the **for** loop adds the numbers from 1 to 100:

```
var sum = 0;  
    for (var i=1; i<=100; i++) {  
        sum = sum + i;  
    }
```

See also

[++ \(increment\)](#)

[-- \(decrement\)](#)

[for..in var](#)

[CONTENTS](#) 

for..in

Syntax

```
for(variableiterant in object){  
statement;  
}
```

Arguments `variableiterant` The name of a variable to act as the iterant, referencing each property of an object or element in an array.

`object` The name of an object to be iterated over.

`statement` A statement to execute for each iteration.

Description Action; loops through the properties of an object or element in an array, and executes the `statement` for each property of an object.

Some properties can not be enumerated by the `for` or `for..in` actions. For example, the built-in methods of the Array object (`Array.sort` and `Array.reverse`) are not included in the enumeration of an Array object, and movie clip properties such as `_x` and `_y` are not enumerated.

The `for..in` construct iterates over properties of objects in the iterated object's prototype chain. If the child's prototype is parent, iterating over the properties of the child with `for..in`, will also iterate over the properties of parent.

Player Flash 5 or later.

Example The following is an example of using `for..in` to iterate over the properties of an object:

```
myObject = { name:'Tara', age:27, city:'San Francisco' };for (name  
in myObject) { trace ("myObject." + name + " = " + myObject[name]);  
}
```

The output of this example is as follows:

```
myObject.name = Tara myObject.age = 27 myObject.city = San Francisco
```

The following is an example of using the `typeof` operator with `for..in` to iterate over a particular type of child:

```
for (name in myMovieClip) { if (typeof (myMovieClip[name]) =  
"movieclip") { trace ("I have a movie clip child named " + name); }  
}
```

The following example enumerates the children of a movie clip and sends each to frame 2 in their respective Timelines. The `RadioButtonGroup` movie clip is a parent with several children, `_RedRadioButton`, `_GreenRadioButton` and `_BlueRadioButton`.

```
for (var name in RadioButtonGroup) {  
RadioButtonGroup[name].gotoAndStop(2); }
```

[CONTENTS](#) 

fscommand

Syntax

```
fscommand(command, arguments);
```

Arguments `command` A string passed to the host application for any use.

`arguments` A string passed to the host application for any use.

Description Action; allows the Flash movie to communicate with the program hosting the Flash Player. In a Web browser, fscommand calls the JavaScript function `moviename_Dofsccommand` in the HTML page containing the Flash movie, where `moviename` is the name of the Flash Player as assigned by the NAME attribute of the EMBED tag or the ID property of the OBJECT tag. If the Flash Player is assigned the name `theMovie`, the JavaScript function called is `theMovie_Dofsccommand`.

Player Flash 3 or later.

[CONTENTS](#) 

function

Syntax

```
function functionname ([argument0, argument1,...argumentN]){  
    statement(s)  
}  
function ([argument0, argument1,...argumentN]){  
    statement(s)  
}
```

Arguments `functionname` The name of the new function.

`argument` Zero or more strings, numbers, or objects to pass the function.

`statements` Zero or more ActionScript statements you have defined for the body of the function.

Description Action; a set of statements that you define to perform a certain task. You can declare, or define, a function in one location and call, or invoke, it from different scripts in a movie. When you define a function, you can also specify arguments for the function. Arguments are placeholders for values on which the function will operate. You can pass a function different arguments, also called parameters, each time you call it.

Use the return action in a functions `statement(s)` to cause a function to return, or generate, a value.

Usage 1: Declares a function with the specified `functionname`, `arguments`, and `statement(s)`. When a function is called, the function declaration is invoked. Forward referencing is permitted; within the same Action list, a function may be declared after it is called. A function declaration replaces any prior declaration of the same function. You can use this syntax wherever a statement is permitted.

Usage 2: Creates an anonymous function and returns it. This syntax is used in expressions, and is particularly useful for installing methods in objects.

Player Flash 5 or later.

Example (Usage 1) The following example defines the function `sqr`, which accepts one argument, and returns the square($x \times x$) of the argument. Note that if the function is declared and used in the same script, the function declaration may appear after using the function.

```
y=sqr(3);  
function sqr(x) {  
    return x*x;  
}
```

(Usage 2) The following function defines a Circle object:

```
function Circle(radius) {  
    this.radius = radius;  
}
```

The following statement defines an anonymous function that calculates the area of a circle and attaches it to the object Circle as a method:

```
Circle.prototype.area = function () {return Math.PI * this.radius * this.radius}
```

[CONTENTS](#) 

getURL

Syntax

```
getURL(url[,window [, variables]]);
```

Arguments `url` The URL from which to obtain the document. The URL must be in the same subdomain as the URL where the movie currently resides.

`window` An optional argument specifying the window or HTML frame that the document should be loaded into. Enter the name of a specific window or choose from the following reserved target names:

- ▶ `_self` specifies the current frame in the current window.
- ▶ `_blank` specifies a new window.
- ▶ `_parent` specifies the parent of the current frame.
- ▶ `_top` specifies the top-level frame in the current window.

`variables` An optional argument specifying a method for sending variables. If there are no variables, omit this argument; otherwise, specify whether to load variables using a GET or POST method. GET appends the variables to the end of the URL, and is used for small numbers of variables. POST sends the variables in a separate HTTP header and is used for long strings of variables.

Description Action; loads a document from a specific URL into a window, or passes variables to another application at a defined URL. To test this action, make sure the file to be loaded is at the specified location. To use an absolute URL (for example, <http://www.myserver.com>), you need a network connection.

Player Flash 2 or later. The GET and POST options are only available to Flash 4 and later versions of the Player.

Example This example loads a new URL into a blank browser window. The `getURL` action targets the variable `incomingAd` as the `url` parameter so that you can change the loaded URL without having to edit the Flash movie. The `incomingAd` variable's value is passed into Flash earlier in the movie using a `loadVariables` action.

```
on(release) {  
    getURL(incomingAd, "_blank");  
}
```

See also

[loadVariables XML.send XML.sendAndLoad XMLSocket.send](#)

CONTENTS 

gotoAndPlay

Syntax

```
gotoAndPlay(scene, frame);
```

Arguments `scene` The scene name to which the playhead is sent.

`frame` The frame number to which the playhead is sent.

Description Action; sends the playhead to the specified frame in a scene and plays from that frame. If no scene is specified, the playhead goes to the specified frame in the current scene.

Player Flash 2 or later.

Example When the user clicks a button that the gotoAndPlay action is assigned to, the playhead is sent to frame 16 and starts to play.

```
on(release) {  
    gotoAndPlay(16);  
}
```

[CONTENTS](#) 

gotoAndStop

Syntax

```
gotoAndStop(scene, frame);
```

Arguments `scene` The scene name to which the playhead is sent.

`frame` The frame number to which the playhead is sent.

Description Action; sends the playhead to the specified frame in a scene and stops it. If no scene is specified, the playhead is sent to the frame in the current scene.

Player Flash 2 or later.

Example When the user clicks a button that the gotoAndStop action is assigned to, the playhead is sent to frame 5 and the movie stops playing.

```
on(release) {  
    gotoAndStop(5);  
}
```

[CONTENTS](#) 

if

Syntax

```
if(condition) {  
    statement;  
}
```

Arguments conditional An expression that evaluates to true or false. For example, `if(name == "Erica")`, evaluates the variable name to see if it is "Erica."

statements The instructions to execute if or when the condition evaluates to true.

Description Action; evaluates a condition to determine the next action in a movie. If the condition is true, Flash runs the statements that follow. Use `if` to create branching logic in your scripts.

Player Flash 4 or later.

See also

[else](#)
[for](#)
[for..in](#)

[CONTENTS](#) 

ifFrameLoaded

Syntax

```
ifFrameLoaded(scene, frame) {  
statement;}  
ifFrameLoaded(frame) {  
statement;}
```

Arguments scene The scene that is being queried.

frame The frame number or frame label to load before the next statement is executed.

Description Action; checks whether the contents of a specific frame are available locally. Use ifFrameLoaded to start playing a simple animation while the rest of the movie downloads to the local computer. The difference between using _framesloaded and ifFrameLoaded is that _framesloaded allows you to add if, or else statements, while ifFrameLoaded action allows you to specify a number of frames in one simple statement.

Player Flash 3 or later. The ifFrameLoaded action is deprecated in Flash 5; use of the _framesloaded action is encouraged.

See also

[_framesloaded](#)

CONTENTS 

#include

Syntax

```
#include "filename.as";
```

Arguments filename.as The filename to include; .as is the recommended file extension.

Description Action; includes the contents of the file specified in the argument when the movie is tested, published, or exported. The #include action is invoked when you test, publish, or export. The #include action is checked when a syntax check occurs.

Player N/A

[CONTENTS](#) 

loadMovie

Syntax

```
loadMovie(url[,location/target, variables]);
```

Arguments **url** An absolute or relative URL for the SWF file to load. A relative path must be relative to the SWF. The URL must be in the same subdomain as the URL where the movie currently resides. For use in the Flash Player or for testing in test-movie mode in the Flash authoring environment, all SWF files must be stored in the same folder, and the file names cannot include folder or disk drive specifications.

target An optional argument specifying a target movie clip that will be replaced by the loaded movie. The loaded movie inherits the position, rotation, and scale properties of the targeted movie clip. Specifying a **target** is the same as specifying the **location** (level) of a target movie; you should not specify both.

location An optional argument specifying the level into which the movie is loaded. The loaded movie inherits the position, rotation, and scale properties of the targeted movie clip. To load the new movie in addition to existing movies, specify a level that is not occupied by another movie. To replace an existing movie with the loaded movie, specify a level that is currently occupied by another movie. To replace the original movie and unload every level, load the new movie into level 0. The movie in level 0 sets the frame rate, background color, and frame size for all other loaded movies.

variables An optional argument specifying a method for sending variables associated with the movie to load. The argument must be the string "GET" or "POST." If there are no variables, omit this argument; otherwise, specify whether to load variables using a GET or POST method. GET appends the variables to the end of the URL, and is used for small numbers of variables. POST sends the variables in a separate HTTP header and is used for long strings of variables.

Description Action; plays additional movies without closing the Flash Player. Normally, the Flash Player displays a single Flash Player movie (SWF file) and then closes. The loadMovie action lets you display several movies at once or switch between movies without loading another HTML document.

You can load movies into level that already have SWF files loaded. If you do, the new movie will replace the existing SWF file. If you load a new movie into Level 0, every level is unloaded, and Level 0 is replaced with the new file. Use the loadVariables action to keep the active movie, and update the variables with new values.

Use the unloadMovie action to remove movies loaded with the loadMovie action.

Player Flash 3 or later.

Example This loadMovie statement is attached to a navigation button labeled Products. There is an invisible movie clip on the Stage with the instance name dropZone. The loadMovie action uses this movie clip as the target parameter to load the products in the SWF file, into the correct position on the Stage:

```
on(release) {  
    loadMovie("products.swf",_root.dropZone);  
}
```

See also

[unloadMovie _level](#)

CONTENTS 

loadVariables

Syntax

```
loadVariables (url,location [, variables]);
```

Arguments `url` An absolute or relative URL where the variables are located. The host for the URL must be in the same subdomain as the movie when accessed using a Web browser.

`location` A level or target to receive the variables. In the Flash Player, movie files are assigned a number according to the order in which they were loaded. The first movie loads into the bottom level (level 0). Inside the loadMovie action, you must specify a level number for each successive movie. This argument is optional.

`variables` An optional argument specifying a method for sending variables. If there are no variables, omit this argument; otherwise, specify whether to load variables using a GET or POST method. GET appends the variables to the end of the URL and is used for small numbers of variables. POST sends the variables in a separate HTTP header and is used for long strings of variables.

Description Action; reads data from an external file, such as a text file or text generated by a CGI script, Active Server Pages (ASP), or Personal Home Page (PHP), and sets the values for variables in a movie or movie clip. This action can also be used to update variables in the active movie with new values.

The text at the specified URL must be in the standard MIME format `application/x-www-urlformencoded` (a standard format used by CGI scripts). The movie and the variables to be loaded must reside at the same subdomain. Any number of variables can be specified. For example, the phrase below defines several variables:

```
company=Macromedia&address=600+Townsend&city=San+Francisco&zip=94103
```

Player Flash 4 or later.

Example This example loads information from a text file into text fields in the main Timeline (level 0). The variable names of the text fields must match the variable names in the data.txt file.

```
on(release) {  
    loadVariables("data.txt", 0);  
}
```

See also

[getURL](#) [MovieClip.loadMovie](#) [MovieClip.loadVariables](#)

CONTENTS 

nextFrame

Syntax

```
nextFrame( ) ;
```

Arguments None.

Description Action; sends the playhead to the next frame and stops it.

Player Flash 2 or later.

Example When the user clicks a button that a nextFrame action is assigned to, the playhead is sent to the next frame.

```
on (release) {  
    nextFrame(5) ;  
}
```

[CONTENTS](#) 

nextScene

Syntax

```
nextScene ( ) ;
```

Arguments None.

Description Action; sends the playhead to frame 1 of the next scene and stops it.

Player Flash 2 or later.

Example This action is assigned to a button that, when pressed and released, sends the playhead to frame 1 of the next scene.

```
on(release) {  
    nextScene ( ) ;  
}
```

[CONTENTS](#) 

on(mouseEvent)

Syntax

```
on(mouseEvent) {  
    statement;  
}
```

Arguments `statement` The instructions to execute when the `mouseEvent` takes place.

A `mouseEvent` action can have one of the following arguments:

- ▶ **press** The mouse button is pressed while the pointer is over the button.
- ▶ **release** The mouse button is released while the pointer is over the button.
- ▶ **releaseOutside** The mouse button is released while the pointer is outside the button.
- ▶ **rollOver** The mouse pointer rolls over the button.
- ▶ **rollOut** The pointer rolls outside of the button area.
- ▶ **dragOver** While the pointer is over the button, the mouse button has been pressed while rolled outside the button, and then rolled back over the button.
- ▶ **dragOut** While the pointer is over the button, the mouse button is pressed and then rolls outside the button area.
- ▶ **keyPress** ("key") The specified `key` is pressed. The `key` portion of the argument is specified using any of the key codes listed in the Appendix B, "Keyboard Keys and Key Code Values," or any of the key constants listed in the [Property summary for the Key object](#).

Description Handler; specifies the mouse event, or keypress that trigger an action.

Player Flash 2 or later.

Example In the following script, the `startDrag` action executes when the mouse is pressed and the conditional script is executed when the mouse is released and the object is dropped:

```
on(press) {  
    startDrag("rabbit");  
}  
on(release) {  
    if(getproperty("", _droptarget) == target) {  
        setProperty ("rabbit", _x, _root.rabbit_x);  
        setProperty ("rabbit", _y, _root.rabbit_y);  
    } else {  
_root.rabbit_x = getProperty("rabbit", _x);  
_root.rabbit_y = getProperty("rabbit", _y);  
_root.target = "pasture";  
    }  
    trace(_root.rabbit_y);  
    trace(_root.rabbit_x);  
    stopDrag();  
}
```

See also

[Key \(object\)](#)

[onClipEvent](#)

[CONTENTS](#) 

onClipEvent

Syntax

```
onClipEvent(movieEvent);{  
    ...  
}
```

Arguments A `movieEvent` is a trigger event that executes actions that are assigned to a movie clip instance. Any of the following values can be specified for the `movieEvent` argument:

- ▶ `load` The action is initiated as soon as the movie clip is instantiated and appears in the Timeline.
- ▶ `unload` The action is initiated in the first frame after the movie clip is removed from the Timeline. The actions associated with the `Unload` movie clip event are processed before any actions are attached to the affected frame.
- ▶ `enterFrame` The action is initiated as each frame is played, similar to actions attached to a movie clip. The actions associated with the `OnEnterFrame` movie clip event are processed after any actions that are attached to the affected frames.
- ▶ `mouseMove` The action is initiated every time the mouse is moved. Use the `_xmouse` and `_ymouse` properties to determine the current mouse position.
- ▶ `mouseDown` The action is initiated when the left mouse button is pressed.
- ▶ `mouseUp` The action is initiated when the left mouse button is released.
- ▶ `keyDown` The action is initiated when a key is pressed. Use the `Key.getCode` method to retrieve information about the last key pressed.
- ▶ `keyUp` The action is initiated when a key is released. Use the `Key.getCode` method to retrieve information about the last key pressed.
- ▶ `data` The action is initiated when data is received in a `loadVariables` or `loadMovie` action. When specified with a `loadVariables` action, the `data` event occurs only once, when the last variable is loaded. When specified with a `loadMovie` action, the `data` event occurs repeatedly, as each section of data is retrieved.

Description Handler; triggers actions defined for a specific instance of a movie clip.

Player Flash 5 or later.

Example The following statement includes the script from an external file when the movie clip instance is loaded and first appears on the Timeline:

```
onClipEvent(load) {  
    #include "myScript.as"  
}
```

The following example uses `onClipEvent` with the `keyDown` movie event. The `keyDown` movie event is usually used in conjunction with one or more methods and properties associated with the `Key` object. In the script below, `key.getCode` is used to find out which key the user has pressed; the returned value is associated with the `RIGHT` or `LEFT` `Key` object properties, and the movie is directed accordingly.

```
onClipEvent(keyDown) { if (Key.getCode() == Key.RIGHT) { }  
_parent.nextFrame(); else if (Key.getCode() == Key.LEFT) {  
_parent.prevFrame(); }
```

The following example uses `onClipEvent` with the `mouseMove` movie event. The `xmouse` and `ymouse` properties track the position of the mouse.

```
onClipEvent(mouseMove) { stageX=_root.xmouse; stageY=_root.ymouse; }
```

See also

[on\(mouseEvent\)](#)

[Key \(object\)](#)

[_xmouse](#)

[_ymouse](#)

[CONTENTS](#) 

play

Syntax

```
play( ) ;
```

Arguments None.

Description Action; moves the playhead forward in the Timeline.

Player Flash 2 or later.

Example The following code uses an if statement to check the value of a name the user enters. If the user enters Steve, the play action is called and the playhead moves forward in the Timeline. If the user enters anything other than Steve, the movie does not play and a text field with the variable name alert is displayed.

```
stop();  
if (name = "Steve") {  
    play();  
} else {  
    alert = "You are not Steve!";  
}
```

[CONTENTS](#) 

prevFrame

Syntax

```
prevFrame( ) ;
```

Arguments None.

Description Action; sends the playhead to the previous frame and stops it.

Player Flash 2 or later.

Example When the user clicks a button that a prevFrame action is assigned to, the playhead is sent to the previous frame.

```
on(release) {  
    prevFrame(5) ;  
}
```

See also

[MovieClip.prevFrame](#)

[CONTENTS](#) 

prevScene

Syntax

```
prevScene ( ) ;
```

Arguments None.

Description Action; sends the playhead to frame 1 of the previous scene and stops it.

Player Flash 2 or later.

See also

[nextScene](#)

[CONTENTS](#) 

print

Syntax

```
print (target, "bmovie");  
print (target, "bmax");  
print (target, "bframe");
```

Arguments **target** The instance name of movie clip to print. By default, all of the frames in the movie are printed. If you want to print only specific frames in the movie, designate frames for printing by attaching a #P frame label to those frames in the authoring environment.

bmovie Designates the bounding box of a specific frame in a movie as the print area for all printable frames in the movie. Attach a #b label (in the authoring environment) to designate the frame whose bounding box you want to use as the print area.

bmax Designates a composite of all of the bounding boxes, of all the printable frames, as the print area. Specify the **bmax** argument when the printable frames in your movie vary in size.

bframe Designates that the bounding box of each printable frame be used as the print area for that frame. This changes the print area for each frame and scales the objects to fit the print area. Use **bframe** if you have objects of different sizes in each frame and want each object to fill the printed page.

Description Action; prints the **target** movie clip according to the printer modifier specified in the argument. If you want to print only specific frames in the target movie, attach a #P frame label to the frames you want to print. Although the print action results in higher quality prints than the printAsBitmap action, it cannot be used to print movies that use alpha transparencies or special color effects.

If you do not specify a print area argument, the print area is determined by the Stage size of the loaded movie by default. The movie does not inherit the main movie's Stage size. You can control the print area by specifying the **bmovie**, **bmax**, or **bframe** arguments.

All of the printable elements in a movie must be fully loaded before printing can begin.

The Flash Player printing feature supports PostScript and non-PostScript printers. Non-PostScript printers convert vectors to bitmaps.

Player Flash 5 or later.

Example The following example will print all of the printable frames in myMovie with the print area defined by the bounding box of the frame with the #b frame label attached:

```
print( "myMovie", "bmovie" );
```

The following example will print all of the printable frames in myMovie with a print area defined by the bounding box of each frame:

```
print( "myMovie", "bframe" );
```

See also

[printAsBitmap](#)

[CONTENTS](#) 

printAsBitmap

Syntax

```
printAsBitmap(target,"bmovie");  
printAsBitmap(target,"bmax");  
printAsBitmap(target,"bframe");
```

Arguments `target` The instance name of movie clip to print. By default, all of the frames in the movie are printed. If you want to print only specific frames in the movie, designate frames for printing by attaching a #P frame label to those frames in the authoring environment.

`bmovie` Designates the bounding box of a specific frame in a movie as the print area for all printable frames in the movie. Attach a #b label (in the authoring environment) to designate the frame whose bounding box you want to use as the print area.

`bmax` Designates a composite of all of the bounding boxes, of all the printable frames, as the print area. Specify the `bmax` argument when the printable frames in your movie vary in size.

`bframe` Designates that the bounding box of each printable frame be used as the print area for that frame. This changes the print area for each frame and scales the objects to fit the print area. Use `bframe` if you have objects of different sizes in each frame and want each object to fill the printed page.

Description Action; prints the `target` movie clip as a bitmap. Use `printAsBitmap` to print movies that contain frames with objects that use transparency or color effects. The `printAsBitmap` action prints at the highest available resolution of the printer in order to maintain as much definition and quality as possible. To calculate the printable file size of a frame designated to print as a bitmap, multiply pixel width by pixel height by printer resolution.

If your movie does not contain alpha transparencies or color effects, it is recommended that you use the `print` action for better quality results.

By default, the print area is determined by the Stage size of the loaded movie. The movie does not inherit the main movie's Stage size. You can control the print area by specifying the `bmovie`, `bmax`, or `bframe` arguments.

All of the printable elements in a movie must be fully loaded before printing can begin.

The Flash Player printing feature supports PostScript and non-PostScript printers. Non-PostScript printers convert vectors to bitmaps.

Player Flash 5 or later.

See also

[print](#)

[CONTENTS](#) 

return

Syntax

```
return[expression];  
return;
```

Arguments *expression* A type, string, number, array, or object to evaluate and return as a value of the function. This argument is optional.

Description Action; specifies the value returned by a function. When the return action is executed, the *expression* is evaluated and returned as a value of the function. The return action causes the function to stop executing. If the return statement is used alone, or if Flash does not encounter a return statement during the looping action, it returns null.

Player Flash 5 or later.

Example The following is an example of using return:

```
function sum(a, b, c){ return a + b + c; }
```

See also

[function](#)

[CONTENTS](#) 

set

Syntax

```
variable = expression;  
set(variable, expression);
```

Arguments *variable* The name of the container that holds the value of the *expression* argument.

expression The value (or a phrase that can be evaluated to a value) that is assigned to the variable.

Description Action; assigns a value to a variable. A variable is a container that holds information. The container itself is always the same, but the contents can change. By changing the value of a variable as the movie plays, you can record and save information about what the user has done, record values that change as the movie plays, or evaluate whether a condition is true or false.

Variables can hold either numbers or strings of characters. Each movie and movie clip has its own set of variables, and each variable has its own value independent of variables in other movies or movie clips.

ActionScript is an untyped language. That means that variables do not need to be explicitly defined as containing either a number or a string. Flash interprets the data type as an integer or string accordingly.

Use the set statement in conjunction with the call action to pass or return values.

Player Flash 4 or later.

Example This example sets a variable called `orig_x_pos` that stores the original x axis position of the ship movie clip in order to reset the ship to its starting location later in the movie:

```
on(release) {  
    set(x_pos, getProperty ("ship", _x ));  
}
```

This is equivalent to writing the following:

```
on(release) {  
    orig_x_pos = getProperty ("ship", _x );  
}
```

See also

[var call](#)

[CONTENTS](#) 

setProperty

Syntax

```
setProperty(target, property, expression);
```

Arguments *target* The path to the instance name of the movie clip whose property is being set.

property The property to be set.

expression The value to which the property is set.

Description Action; changes the property of a movie clip as the movie plays.

Player Flash 4 or later.

Example This statement sets the `_alpha` property of a movie clip named `star` to 30 percent when the button is clicked:

```
on(release) {  
    setProperty("star", _alpha = 30);  
}
```

See also

[getProperty](#)

CONTENTS 

Sound.stop

Syntax

```
mySound.stop( ) ;  
mySound.stop( [ "idName" ] ) ;
```

Arguments `idName` An optional argument specifying a specific sound to stop playing. The `idName` argument must be enclosed in quotation marks(" ").

Description Method; stops all sounds currently playing if no argument is specified, or just the sound specified in the `idName` argument.

Player Flash 5 or later.

See also

[Sound.start](#)

[CONTENTS](#) 

stopAllSounds

Syntax

```
stopAllSounds( ) ;
```

Arguments None.

Description Action; stops all sounds currently playing in a movie without stopping the playhead. Sounds set to stream will resume playing as the playhead move over the frames they are in.

Player Flash 3 or later.

Example The following code could be applied to a button that, when clicked, stops all sounds in the movie:

```
on(release) {  
    stopAllSounds( ) ;  
}
```

See also

[Sound \(object\)](#)

[CONTENTS](#) 

tellTarget

Syntax

```
tellTarget(target) {  
    statement;  
}
```

Arguments *target* A target path string specifying the Timeline to be controlled.

statement Instructions applied to the targeted Timeline.

Description Action; applies the instructions specified in the *statements* argument to the Timeline specified in the *target* argument. The tellTarget action is useful for navigation controls. Assign tellTarget to buttons that stop or start movie clips elsewhere on the Stage. You can also make movie clips go to a particular frame in that clip. For example, you might assign tellTarget to buttons that stop or start movie clips on the Stage or prompt movie clips to jump to a particular frame.

The tellTarget action is very similar to the with action, except that with takes a movie clip or other object as a *target*, and tellTarget requires a target path to a movie clip and cannot control objects.

Player Flash 3 or later. This action is deprecated in Flash 5; use of the with action is recommended.

Example This tellTarget statement controls the movie clip instance ball on the main Timeline. Frame 1 of the movie clip is blank and has a stop action so that it isn't visible on the Stage. When the button with the following action is clicked, tellTarget tells the playhead in the movie clip ball to go to frame 2 and play the animation that starts there.

```
on(release) {  
    tellTarget("ball") {  
        gotoAndPlay(2);  
    }  
}
```

See also

[with](#)

[CONTENTS](#) 

toggleHighQuality

Syntax

```
toggleHighQuality();
```

Arguments None.

Description Action; turns antialiasing on and off in the Flash Player. Antialiasing smooths the edges of objects and slows down the movie playback. The toggleHighQuality action affects all movies in the Flash Player.

Player Flash 2 or later.

Example The following code could be applied to a button that, when clicked, would toggle antialiasing on and off:

```
on(release) {  
    toggleHighQuality();  
}
```

See also

[_quality_highquality](#)

[CONTENTS](#) 

trace

Syntax

```
trace(expression);
```

Arguments `expression` A statement to evaluate. When you test the movie, the results of the `expression` argument are displayed in the Output window.

Description Action; evaluates the `expression` and displays the results in the Output window in test-movie mode.

Use trace to record programming notes or to display messages in the Output window while testing a movie. Use the `expression` parameter to check if a condition exists, or to display values in the Output window. The trace action is similar to the alert function in JavaScript.

Player Flash 4 or later.

Example This example is from a game in which a draggable movie clip instance named `rabbi` must be released on a specific target. A conditional statement evaluates the `_droptarget` property and executes different actions depending on where `rabbi` is released. The trace action is used at the end of the script to evaluate the location of the `rabbi` movie clip, and display the results in the Output window. If `rabbi` doesn't behave as expected (for example, if it snaps to the wrong target), the values sent to the Output window by the trace action will help you determine the problem in the script.

```
on(press) { rabbi.startDrag(); } on(release) { if(eval(_droptarget)
!= target) { rabbi._x = rabbi_x; rabbi._y = rabbi_y; } else { rabbi_x
= rabbi._x; rabbi_y = rabbi._y; target = "_root.pasture"; }
trace("rabbi_y = " + rabbi_y); trace("rabbi_x = " + rabbi_x);
stopDrag(); }
```

[CONTENTS](#) 

var

Syntax

```
var variableName1 [=value1] [...,variableNameN [=valueN]];
```

Arguments `variableName` The name of the variable to declare.

`value` The value being assigned to the variable.

Description Action; used to declare local variables. If you declare local variables inside a function, the variables are defined for the function and expire at the end of the function call. If variables are not declared inside a block, but the action list was executed with a call action, the variables are local and expire at the end of the current list. If variables are not declared inside a block and the current action list was not executed with the call action, the variables are not local.

Player Flash 5 or later.

[CONTENTS](#) 

while

Syntax

```
while(condition) {  
    statement(s);  
}
```

Arguments **condition** The statement that is reevaluated each time the while action is executed. If the statement evaluates to true, the expression in the **statement(s)** is run.

statement(s) The expression to run if the condition evaluates to true.

Description Action; runs a statement or series of statements repeatedly in a loop as long as the condition argument is true. At the end of each while action, Flash restarts the loop by retesting the condition. If the condition is false or equal to 0, Flash skips to the first statement after the while action.

Looping is commonly used to perform an action while a counter variable is less than a specified value. At the end of each loop, the counter is incremented until the threshold value is reached, the **condition** is no longer true, and the loop ends.

Player Flash 4 or later.

Example This example duplicates five movie clips on the Stage, each with a randomly generated x and y position, xscale and yscale, and _alpha property to achieve a scattered effect. The variable `foo` is initialized with the value 0. The **condition** argument is set so that the while loop will run five times, or as long as the value of the variable `foo` is less than 5. Inside the while loop, a movie clip is duplicated and `setProperty` is used to adjust the various properties of the duplicated movie clip. The last statement of the loop increments `foo` so that when the value reaches 5, the **condition** argument evaluates to false, and the loop will not be executed.

```
on(release) {  
    foo = 0;  
    while(foo < 5) {  
        duplicateMovieClip("/flower", "mc" + foo, foo);  
        setProperty("mc" + foo, _x, random(275));  
        setProperty("mc" + foo, _y, random(275));  
        setProperty("mc" + foo, _alpha, random(275));  
        setProperty("mc" + foo, _xscale, random(200));  
        setProperty("mc" + foo, _yscale, random(200));  
        foo = foo + 1;  
    }  
}
```

See also

[do... while](#)
[continue](#)

[CONTENTS](#) 

with

Syntax

```
with (object){  
    statement(s);  
}
```

Arguments `object` An instance of an ActionScript object or movie clip.

`statement(s)` An action or group of actions enclosed in curly braces.

Description Action; temporarily changes the scope (or target path) used for evaluating expressions and actions in the `statement(s)`. After the `with` action executes, the scope chain is restored to its original state.

The `object` becomes the context in which the properties, variables, and functions are read. For example, if `object` is `myArray`, and two of the properties specified are `length` and `concat`, those properties are automatically read as `myArray.length` and `myArray.concat`. In another example, if `object` is `state.california`, it is as if any actions or statements inside the `with` action were called from inside the `california` instance.

To find the value of an identifier in the `statement(s)`, ActionScript starts at the beginning of the scope chain specified by the `object` and searches for the identifier at each level of the scope chain, in a specific order.

The scope chain used by the `with` action to resolve identifiers starts with the first item in the following list and continues to the last, as follows:

- ▶ `object` referenced by innermost `with` action
- ▶ `object` referenced by outermost `with` action
- ▶ Activation object (A temporary object that is automatically created when a function is called that holds the local variables called in the function.)
- ▶ Movie clip containing currently executing script
- ▶ Global object (predefined objects such as `Math`, `String`)

In Flash 5 the `with` action replaces the deprecated `tellTarget` action. You are encouraged to use `with` instead of `tellTarget` because it is a standard ActionScript extension to the ECMA-262 standard. The principal difference between the `with` and `tellTarget` actions is that `with` takes a reference to a movie clip or other object as its argument, while `tellTarget` takes a target path string identifying a movie clip, and cannot be used to target objects.

To set a variable inside a `with` action, the variable must have been declared outside the `with` action or you must enter the full path to the Timeline on which you want the variable to live. If you set a variable in a `with` action without having declared it, the `with` action will look for the value according to the scope chain. If the variable doesn't already exist, the new value will be set on the Timeline from which the `with` action was called.

Example The following example sets the `x` and `y` properties of the `someOtherMovieClip` instance, and then instructs `someOtherMovieClip` to go to frame 3 and stop:

```
with (someOtherMovieClip) {  
    _x = 50;  
    _y = 100;  
    gotoAndStop(3);  
}
```

The following code snippet is how you would write the preceding code without using a `with` action:

```
someOtherMovieClip._x = 50;  
someOtherMovieClip._y = 100;  
someOtherMovieClip.gotoAndStop(3);
```

This code could also be written using the `tellTarget` action:

```
tellTarget ("someOtherMovieClip") {  
    _x = 50;  
    _y = 100;
```

```
        gotoAndStop(3);  
    }
```

The `with` action is useful for accessing multiple items in a scope chain list simultaneously. In the following example, the built-in `Math` object is placed at the front of the scope chain. Setting `Math` as a default object resolves the identifiers `cos`, `sin`, and `PI` to `Math.cos`, `Math.sin`, and `Math.PI`, respectively. The identifiers `a`, `x`, `y`, and `r` are not methods or properties of the `Math` object, but since they exist in the object activation scope of the function `polar`, they resolve to the corresponding local variables.

```
function polar(r){  
    var a, x, y  
    with (Math) {  
        a = PI * r * r  
        x = r * cos(PI)  
        y = r * sin(PI/2)  
    }  
    trace("area = " + a)  
    trace("x = " + x)  
    trace("y = " + y)  
}
```

You can use nested `with` actions to access information in multiple scopes. In the following example, the instance `fresno` and the instance `salinas` are children of the instance `california`. The statement sets the `_alpha` values of `fresno` and `salinas` without changing the `_alpha` value of `california`.

```
with (california){  
    with (fresno){  
        _alpha = 20;  
    }  
    with (salinas){  
        _alpha = 40;  
    }  
}
```

See also

[tellTarget](#)

CONTENTS 

Boolean (function)

Syntax

```
Boolean(expression) ;
```

Arguments `expression` The variable, number, or string to be converted to a Boolean.

Description Function; converts the specified argument to a Boolean, and returns the Boolean value.

Player Flash 5 or later.

[CONTENTS](#) 

chr

Syntax

```
chr ( number );
```

Arguments `number` The ASCII code number to convert to a character.

Description String function; converts ASCII code numbers to characters.

Player Flash 4 or later. This function has been deprecated in Flash 5; use of the `String.fromCharCode` method is recommended.

Example The following example converts the number 65 to the letter "A":

```
chr ( 65 ) = "A"
```

See also

[String.fromCharCode](#)

CONTENTS 

escape

Syntax

```
escape ( expression ) ;
```

Arguments `expression` The expression to convert into a string and encode in a URL-encoded format.

Description Function; converts the argument to a string and encodes it in a URL-encoded format, where all alphanumeric characters are escaped with % hexadecimal sequences.

Player Flash 5 or later.

Example `escape("Hello{[World]}")`;

The result of the above code is as follows:

Hello%7B%5BWorld%5D%7D

See also

[unescape](#)

CONTENTS 

eval

Syntax

```
eval(expression);
```

Arguments `expression` A string containing the name of a variable, property, object or movie clip to retrieve.

Description Function; accesses variables, properties, objects, or movie clip by name. If the `expression` is a variable or a property, the value of the variable or property is returned. If the `expression` is an object or movie clip, a reference to the object or movie clip is returned. If the element named in the `expression` can not be found, undefined is returned.

In Flash 4, the eval function was used to simulate an arrays. In Flash 5 it is recommended that you use the Array object to create arrays.

Note: The ActionScript eval action is not the same as the JavaScript eval function, and cannot be used to evaluate statements.

Player Flash 5 or later for full functionality. You can use eval when exporting to the Flash 4 Player, but you must use slash notation, and can only access variables, not properties or objects.

Example The following example uses eval to determine the value of the variable x, and sets it to the value of y:

```
x = 3;  
y = eval("x");
```

The following example uses eval to reference the movie clip object associated with a movie clip instance on the Stage, Ball:

```
eval("_root.Ball");
```

See also

[Array \(object\)](#)

[CONTENTS](#) 

evaluate

Syntax `statement;`

Arguments None.

Description Action; creates a new empty line and inserts a `;` for entering unique scripting statements using Expression field in the Actions panel. The `evaluate` statement also allows users who are scripting in the Flash 5 Actions panel's Normal Mode to call functions.

Player Flash 5 or later.

[CONTENTS](#) 

getProperty

Syntax

```
getProperty(instanceName , property);
```

Arguments instanceName The instance name of a movie clip for which the property is being retrieved.

property A property of a movie clip, such as an x or y coordinate.

Description Function; returns the value of the specified property for the movie clip instance.

Player Flash 4 or later.

Example The following example retrieves the horizontal axis coordinate (_x) for the movie clip myMovie:

```
getProperty(_root.myMovie_item._x);
```

[CONTENTS](#) 

getTimer

Syntax

```
getTimer( );
```

Arguments None.

Description Function; returns the number of milliseconds that have elapsed since the movie started playing.

Player Flash 4 or later.

[CONTENTS](#) 

getVersion

Syntax

```
getVersion();
```

Arguments None.

Description Function; returns a string containing Flash Player version and platform information.

This function does not work in test-movie mode, and will only return information for versions 5 or later of the Flash Player.

Example The following is an example of a string returned by the getVersion function:

```
WIN 5,0,17,0
```

This indicates that the platform is Windows, and the version number of the Flash Player is major version 5, minor version 17(5.0r17).

Player Flash 5 or later.

[CONTENTS](#) 

int

Syntax

```
int(value);
```

Arguments `value` A number to be rounded to an integer.

Description Function; converts a decimal number to the closest integer value.

Player Flash 4 or later. This function has been deprecated in Flash 5; use of the `Math.floor` method is recommended.

See also

[Math.floor](#)

CONTENTS 

isFinite

Syntax

```
isFinite(expression);
```

Arguments expression The Boolean, variable, or other expression to be evaluated.

Description Top-level function; evaluates the argument and returns true if it is a finite number, and false if it is infinity or negative infinity. The presence of infinity or negative infinity indicates a mathematical error condition such as division by 0.

Player Flash 5 or later.

Example The following are examples of return values for isFinite:

isFinite(56) returns true

isFinite(Number.POSITIVE_INFINITY) returns false

isNaN(Number.POSITIVE_INFINITY) returns false

[CONTENTS](#) 

isNaN

Syntax

```
isNaN(expression);
```

Arguments expression The Boolean, variable, or other expression to be evaluated.

Description Top-level function; evaluates the argument and returns true if the value is not a number (NaN), indicating the presence of mathematical errors.

Player Flash 5 or later.

Example The following illustrates the return value for isNaN:

isNaN("Tree") returns true

isNaN(56) returns false

isNaN(Number.POSITIVE_INFINITY) returns false

[CONTENTS](#) 

maxscroll

Syntax

```
variable_name.maxscroll = x
```

Arguments `variable_name` The name of a variable associated with a text field.

`x` The line number that is the maximum value allowed for the `scroll` property, based on the height of the text field. This is a read-only value set by Flash.

Description Property; a read-only property that works with the `scroll` property to control the display of information in a text field. This property can be retrieved, but not modified.

Player Flash 4 or later.

See also

[scroll](#)

[CONTENTS](#) 

mbchr

Syntax

`mbchr (number);`

Arguments `number` The number to convert to a multibyte character.

Description String function; converts an ASCII code number to a multibyte character.

Player Flash 4 or later. This function has been deprecated in Flash 5; use of `String.fromCharCode` method is encouraged.

See also

[String.fromCharCode](#)

CONTENTS 

mblength

Syntax

```
mblength(string);
```

Arguments string A string.

Description String function; returns the length of the multibyte character string.

Player Flash 4 or later. This function has been deprecated in Flash 5; use of the String object and methods is recommended.

[CONTENTS](#) 

mbord

Syntax

```
mbord(character);
```

Arguments `character` The character to convert to a multibyte number.

Description String function; converts the specified character to a multibyte number.

Player Flash 4 or later. This function has been deprecated in Flash 5; use of the `String.charCodeAt` method is recommended.

See also

[String.fromCharCode](#)

CONTENTS 

mbsubstring

Syntax

```
mbsubstring(value, index, count);
```

Arguments `value` The multibyte string from which to extract a new multibyte string.

`index` The number of the first character to extract.

`count` The number of characters to include in the extracted string, not including the index character.

Description String function; extracts a new multibyte character string from a multibyte character string.

Player Flash 4 or later. This function is deprecated in Flash 5; use of the `string.substr` method is recommended.

See also

[String.substr](#)

CONTENTS 

newline

Syntax

```
newline;
```

Arguments None.

Description Constant; inserts a carriage return character (`{}`) inserting a blank line into the ActionScript code. Use `newline` to make space for information that is retrieved by a function or action in your code.

Player Flash 4 or later.

[CONTENTS](#) 

Number (function)

Syntax

`Number (expression) ;`

Arguments `expression` The string, Boolean, or other expression to convert to a number.

Description Function; converts the argument `x` to a number and returns a value as follows:

If `x` is a number, the return value is `x`.

If `x` is a Boolean, the return value is 1 if `x` is true, 0 if `x` is false.

If `x` is a string, the function attempts to parse `x` as a decimal number with an optional trailing exponent, that is, 1.57505e-3.

If `x` is undefined, the return value is 0.

This function is used to convert Flash 4 files containing deprecated operators that are imported into the Flash 5 authoring environment. See the `&` operator for more information.

Player Flash 4 or later.

See also

[Number \(object\)](#)

CONTENTS 

ord

Syntax

```
ord(character);
```

Arguments `character` The character to convert to an ASCII code number.

Description String function; converts characters to ASCII code numbers.

Player Flash 4 or later. This function has been deprecated in Flash 5, and it is recommended that you use the methods and properties of the String object instead.

See also

[String \(object\)](#)

[CONTENTS](#) 

parseFloat

Syntax

```
parseFloat(string);
```

Arguments `string` The string to parse and convert to a floating-point number.

Description Function; converts a string to a floating-point number. The function parses and returns the numbers in the string, until the parser reaches a character that is not a part of the initial number. If the string does not begin with a number that can be parsed, parseFloat returns NaN or 0. White space preceding valid integers is ignored, as are trailing non-numeric characters.

Player Flash 5 or later.

Example The following are examples of using parseFloat to evaluate various types of numbers:

parseFloat("-2") returns -2

parseFloat("2.5") returns 2.5

parseFloat("3.5e6") returns 3.5e6, or 3500000

parseFloat("foobar") returns NaN

CONTENTS 

parseInt

Syntax

```
parseInt(expression, radix);
```

Arguments `expression` The string, floating-point number, or other expression to parse and convert to a integer.

`radix` An integer representing the radix (base) of the number to parse. Legal values are from 2 and 36. This argument is optional.

Description Function; converts a string to an integer. If the specified string in the arguments cannot be converted to a number, the function returns NaN or 0. Integers beginning with 0 or specifying a radix of 8 are interpreted as octal numbers. Integers beginning with 0x are interpreted as hexadecimal numbers. White space preceding valid integers is ignored, as are trailing nonnumeric characters.

Player Flash 5 or later.

Example The following are examples of using parseInt to evaluate various types of numbers:

`parseInt("3.5")` returns 3.5

`parseInt("bar")` returns NaN

`parseInt("4foo")` returns 4

Hexadecimal conversion:

`parseInt("0x3F8")` returns 1016

`parseInt("3E8", 16)` returns 1000

Binary conversion:

`parseInt("1010", 2)` returns 10 (the decimal representation of the binary 1010)

Octal number parsing (in this case the octal number is identified by the radix, 8):

`parseInt("777", 8)` returns 511 (the decimal representation of the octal 777)

[CONTENTS](#) 

random

Syntax

```
random( ) ;
```

Arguments `value` The highest integer for which `random` will return a value.

Description Function; returns a random integer between 0 and the integer specified in the `value` argument.

Player Flash 4. This function is deprecated in Flash 5; use of the `Math.random` method is recommended.

Example The following use of `random` returns a value of 0, 1, 2, 3, or 4:

```
random( 5 ) ;
```

See also

[Math.random](#)

[CONTENTS](#) 

scroll

Syntax

```
variable_name.scroll = x
```

Arguments `variable_name` The name of a variable associated with a text field.

`x` The line number of the topmost visible line in the text field. You can specify this value or use the default value of 1. The Flash Player updates this value as the user scrolls up and down the text field.

Description Property; controls the display of information in a text field associated with a variable. The `scroll` property defines where the text field begins displaying content; after you set it, the Flash Player updates it as the user scrolls through the text field. The `scroll` property is useful for directing users to a specific paragraph in a long passage, or creating scrolling text fields. This property can be retrieved and modified.

Player Flash 4 or later.

See also

[maxscroll](#)

CONTENTS 

String (function)

Syntax

```
String(expression) ;
```

Arguments `expression` The number, Boolean, variable, or object to convert to a string.

Description Function; returns a string representation of the specified argument as follows:

If `x` is Boolean, the return string is true or false.

If `x` is a number, the return string is a decimal representation of the number.

If `x` is a string, the return string is `x`.

If `x` is an object, the return value is a string representation of the object generated by calling the string property for the object, or by calling `object.toString` if no such property exists.

If `x` is a movie clip, the return value is the target path of the movie clip in slash (/) notation.

If `x` is undefined, the return value is an empty string.

Player Flash 3 or later.

See also

[Object.toString](#)

[Number.toString](#)

[String \(object\)](#)

[" " \(string delimiter\)](#)

CONTENTS 

targetPath

Syntax

```
targetPath(movieClipObject);
```

Arguments movieClipObject Reference (for example, _root or _parent) to the movie clip for which the target path is being retrieved.

Description Function; returns a string containing the target path of movieClipObject. The target path is returned in dot notation. To retrieve the target path in slash notation, use the _target property.

Player Flash 5 or later.

Example The following examples are equivalent. The first example uses dot notation, and the second example uses slash notation.

```
targetPath (Board.Block[index*2+1]) {  
  
play();  
  
}
```

Is equivalent to:

```
tellTarget ("Board/Block:" + (index*2+1)) {  
  
play();  
  
}
```

See also [eval](#)

CONTENTS 

unescape

Syntax

```
unescape(x) ;
```

Arguments \times A string with hexadecimal sequences to escape.

Description Top-level function; evaluates the argument \times as a string, decodes the string from a URL-encoded format (converting all hexadecimal sequences to ASCII characters), and returns the string.

Player Flash 5 or later.

Example The following example illustrates the escape-to-unescape conversion process.

```
escape( "Hello{[World]}" );
```

The escaped result is as follows:

```
( "Hello%7B%5BWorld%5D%7D" );
```

Use unescape to return to the original format:

```
unescape( "Hello%7B%5BWorld%5D%7D" )
```

The result is as follows:

```
Hello{[World]}
```

[CONTENTS](#) 

updateAfterEvent

Syntax

```
updateAfterEvent(movie clip event);
```

Arguments movie clip event You can specify one of the following values as a movie clip event:

- ▶ **mouseMove** The action is initiated every time the mouse is moved. Use the `__xmouse` and `__ymouse` properties to determine the current mouse position.
- ▶ **mouseDown** The action is initiated if the left mouse button is pressed.
- ▶ **mouseUp** The action is initiated if the left mouse button is released.
- ▶ **keyDown** The action is initiated when a key is pressed. Use the `Key.getCode` method to retrieve information about the last key pressed.
- ▶ **keyUp** The action is initiated when a key is released. Use the `key.getCode` method to retrieve information about the last key pressed.

Description Action; updates the display (independent of the frames per second set for the movie) after the clip event specified in the arguments has completed. This action is not listed in the Flash Actions panel. Using `updateAfterEvent` with drag actions that specify the `_x` and `_y` properties during the mouse move allows objects to drag smoothly without a flickering screen effect.

Player Flash 5 or later.

See also

[onClipEvent](#)

CONTENTS 

Math.abs

Syntax

`Math.abs(x) ;`

Arguments x Any number.

Description Method; computes and returns an absolute value for the number specified by the argument x.

Player Flash 5 or later. In the Flash 4 Player, the methods and properties of the Math object are emulated using approximations and may not be as accurate as the non-emulated math functions supported by the Flash 5 Player.

[CONTENTS](#) 

Math.acos

Syntax

`Math.acos(x);`

Arguments x A number from -1.0 to 1.0.

Description Method; computes and returns the arc cosine of the number specified in the argument x, in radians.

Player Flash 5 or later. In the Flash 4 Player, the methods and properties of the Math object are emulated using approximations and may not be as accurate as the non-emulated math functions supported by the Flash 5 Player.

[CONTENTS](#) 

XML.appendChild

Syntax

```
myXML.appendChild( childNode );
```

Arguments childNode The child node to be added to the specified XML object's child list.

Description Method; appends the specified child node to the XML object's child list. The appended child node is placed in the tree structure once removed from its existing parent node, if any.

Player Flash 5 or later.

Example The following example clones the last node from doc1 and appends it to doc2:

```
doc1 = new XML(src1);  
doc2 = new XML();  
node = doc1.lastChild.cloneNode(true);  
doc2.appendChild(node);
```

[CONTENTS](#) 

Array (object)

x208A1 | IDS_ACTIONHELP_ARRAY_OBJECT, Array object

The Array object allows you to access and manipulate arrays. An array is an object whose properties are identified by a number representing their position in the array. This number is sometimes referred to as the index. All arrays are zero based, which means that the first element in the array is [0], the second element is [1], and so on. In the following example, myArray contains the months of the year, identified by number.

```
myArray[0] = "January"
```

```
myArray[1] = "February"
```

```
myArray[2] = "March"
```

```
myArray[3] = "April"
```

To create an Array object, use the constructor `new Array`. To access the elements of an array use, the array access operator `[]`.

<code>concat</code>	Concatenates the arguments and returns them as a new array.
<code>join</code>	Joins all elements of an array into a string.
<code>pop</code>	Removes the last element of an array, and returns its value.
<code>push</code>	Adds one or more elements to the end of an array and returns the array's new length.
<code>reverse</code>	Reverses the direction of an array.
<code>shift</code>	Removes the first element from an array, and returns its value.
<code>slice</code>	Extracts a section of an array and returns it as a new array.
<code>sort</code>	Sorts an array in place.
<code>splice</code>	Adds and/or removes elements from an array.
<code>toString</code>	Returns a string value representing the elements in the Array object.
<code>unshift</code>	Adds one or more elements to the beginning of an array and returns the array's new length.

Method summary for the Array object

Property summary for the Array object

<code>length</code>	Returns the length of the array.

Constructor for the Array object

Syntax

```
new Array();  
new Array(length);  
new Array(element0, element1, element2,...elementN);
```

Arguments `length` An integer specifying the number of elements in the array. In the case of noncontiguous elements, the `length` specifies the index number of the last element in the array plus 1. For more information, see the property `Array.length`.

`element0...elementN` A list of two or more arbitrary values. The values can be numbers, names, or other elements specified in an array. The first element in an array always has the index or position 0.

Description Constructor; allows you to access and manipulate elements in an array. Arrays are zero based and the elements are indexed by their ordinal number.

If you don't specify any arguments, a zero-length array is created.

Player Flash 5 or later.

Example The following example creates a new Array object with an initial length of 0:

```
myArray = new Array();
```

The following example creates the new Array object A-Team, with an initial length of 4:

```
A-Team = new Array("Jody", "Mary", "Marcelle", "Judy");
```

The initial elements of the A-Team array are as follows:

```
myArray[0] = "Jody"
```

```
myArray[1] = "Mary"
```

```
myArray[2] = "Marcelle"
```

```
myArray[3] = "Judy"
```

See also

[Array.length](#)

[CONTENTS](#) 

Math.asin

Syntax

```
Math.asin(x);
```

Arguments x A number from -1.0 to 1.0.

Description Method; computes and returns the arc sine for the number specified in the argument x, in radians.

Player Flash 5 or later. In the Flash 4 Player, the methods and properties of the Math object are emulated using approximations and may not be as accurate as the non-emulated math functions supported by the Flash 5 Player.

[CONTENTS](#) 

Math.atan

Syntax

`Math.atan(x);`

Arguments x Any number.

Description Method; computes and returns the arc tangent for the number specified in the argument x. The return value is between negative pi divided by 2, and positive pi divided by 2.

Player Flash 5 or later. In the Flash 4 Player, the methods and properties of the Math object are emulated using approximations and may not be as accurate as the non-emulated math functions supported by the Flash 5 Player.

[CONTENTS](#) 

Math.atan2

Syntax

```
Math.atan2(y, x);
```

Arguments x A number specifying the x coordinate of the point.

y A number specifying the y coordinate of the point.

Description Method; computes and returns the arc tangent of y/x in radians. The return value represents the angle opposite the opposite angle of a right triangle, where x is the adjacent side length and y is the opposite side length.

Player Flash 5 or later. In the Flash 4 Player, the methods and properties of the Math object are emulated using approximations and may not be as accurate as the non-emulated math functions supported by the Flash 5 Player.

[CONTENTS](#) 

XML.attributes

Syntax

```
myXML.attributes;
```

Arguments None.

Description Collection (read-write); returns an associative array containing all attributes of the specified XML object.

Player Flash 5 or later.

Example The following example writes the names of the XML attributes to the Output window:

```
str = "<mytag name=\"Val\"> intem </mytage>";  
doc = new XML(str);  
y = doc.firstChild.attributes.name;  
    trace (y);  
doc.firstChild.attributes.order = "first";  
z = doc.firstChild.attributes.order  
    trace(z);
```

The following is written to the Output window:

```
Val  
First
```

[CONTENTS](#) 

Key.BACKSPACE

Syntax Key.BACKSPACE

Arguments None.

Description Property; constant associated with the key code value for the Backspace key (9).

Player Flash 5 or later.

[CONTENTS](#) 

Boolean (object)

The Boolean object is a simple wrapper object with the same functionality as the standard JavaScript Boolean object. Use the Boolean object to retrieve the primitive data type or string representation of Boolean object.

Method summary for the Boolean object

<code>toString</code>	Returns the string representation (<code>true</code>) or (<code>false</code>) of the Boolean object.
<code>valueOf</code>	Returns the primitive value type of the specified Boolean object.

Constructor for the Boolean object

`x208C0` | `IDS_ACTIONHELP_BOOLEAN_NEW`, new Boolean

Syntax

```
new Boolean( ) ;  
new Boolean(x) ;
```

Arguments `x` A number, string, Boolean, object, movie clip, or other expression. This argument is optional.

Description Constructor; creates an instance of the Boolean object. If you omit the `x` argument, the Boolean object is initialized with a value of `false`. If you specify `x`, the method evaluates the argument and returns the result as a Boolean value according to the following casting rules:

- ▶ If `x` is a number, the function returns `true` if `x` does not equal 0, or `false` if `x` is any other number.
- ▶ If `x` is a Boolean, the function returns `x`.
- ▶ If `x` is an object or movie clip, the function returns `true` if `x` does not equal `null`; otherwise, the function returns `false`.
- ▶ If `x` is a string, the function returns `true` if `Number (x)` does not equal 0; otherwise, the function returns `false`.

Note: To maintain compatibility with Flash 4, the handling of strings by the Boolean object is not ECMA-262 standard.

Player Flash 5 or later.

[CONTENTS](#) 

Key.CAPSLOCK

Syntax Key.CAPSLOCK

Arguments None.

Description Property; constant associated with the key code value for the Caps Lock key (20).

Player Flash 5 or later.

[CONTENTS](#) 

Math.ceil

Syntax

```
Math.ceil(x);
```

Arguments x A number or expression.

Description Method; returns the ceiling of the specified number or expression. The ceiling of a number is the closest integer that is greater than or equal to the number.

Player Flash 5 or later. In the Flash 4 Player, the methods and properties of the Math object are emulated using approximations and may not be as accurate as the non-emulated math functions supported by the Flash 5 Player.

[CONTENTS](#) 

String.charAt

Syntax

```
myString.charAt ( index ) ;
```

Arguments `index` The number of the character in the string to be returned.

Description Method; returns the character specified by the argument `index`. The index of the first character in a string is 0. If `index` is not a number from 0 to `string.length - 1`, an empty string is returned.

Player Flash 5 or later.

[CONTENTS](#) 

String.charCodeAtAt

Syntax

```
myString.charCodeAtAt(index);
```

Arguments `index` The number of the character for which the value is retrieved.

Description Method; returns the value of the character specified by `index`. The returned value is a 16-bit integer from 0 to 65535.

This method is similar to `string.charAt` except that the returned value is for the character at a specific location, instead of a substring containing the character.

Player Flash 5 or later.

[CONTENTS](#) 

XML.childNodes

Syntax `myXML.childNodes;`

Arguments None.

Description Collection (read-only); returns an array of the specified XML object's children. Each element in the array is a reference to an XML object that represents a child node. This is a read-only property and cannot be used to manipulate child nodes. Use the methods `appendChild`, `insertBefore`, and `removeNode` to manipulate child nodes.

This collection is undefined for text nodes (`nodeType == 3`).

Player Flash 5 or later.

[CONTENTS](#) 

XML.cloneNode

Syntax `myXML.cloneNode(deep) ;`

Arguments `deep` Boolean value specifying whether the children of the specified XML object are recursively cloned.

Description Method; constructs and returns a new XML node of the same type, name, value, and attributes as the specified XML object. If `deep` is set to `true`, all child nodes are recursively cloned, resulting in an exact copy of the original object's document tree.

Player Flash 5 or later.

[CONTENTS](#) 

XMLSocket.close

Syntax

```
myXMLSocket.close( ) ;
```

Arguments None.

Description Method; closes the connection specified by XMLSocket object.

Player Flash 5 or later.

See also

[XMLSocket.connect](#)

[CONTENTS](#) 

Color (object)

The Color object allows you to set and retrieve the RGB color value and color transform of movie clips. The Color object is supported by Flash 5 and later versions of the Flash Player.

You must use the constructor `new Color()` to create an instance of the Color object before calling the methods of the Color object.

Method summary for the Color object

<code>getRGB</code>	Returns the numeric RGB value set by the last <code>setRGB</code> call.
<code>getTransform</code>	Returns the transform information set by the last <code>setTransform</code> call.
<code>setRGB</code>	Sets the hexadecimal representation of the RGB value for a Color object.
<code>setTransform</code>	Sets the color transform for a Color object.

Constructor for the Color object

Syntax

```
new Color(target);
```

Arguments `target` The name of the movie clip the new color is applied to.

Description Constructor; creates a Color object for the movie clip specified by the `target` argument.

Player Flash 5 or later.

Example The following example creates a new Color object called `myColor` for the movie `myMovie`:

```
myColor = new Color(myMovie);
```

[CONTENTS](#) 

Array.concat

Syntax

```
myArray.concat(value0,value1,...valueN);
```

Arguments value0,...valueN Numbers, elements, or strings to be concatenated in a new array.

Description Method; concatenates the elements specified in the arguments, if any, and creates and returns a new array. If the arguments specify an array, the elements of that array are concatenated, rather than the array itself.

Player Flash 5 or later.

Example The following code concatenates two arrays:

```
alpha = new Array("a","b","c"); numeric = new Array(1,2,3);  
alphaNumeric=alpha.concat(numeric); // creates array  
["a","b","c",1,2,3]
```

The following code concatenates three arrays:

```
num1=[1,3,5]; num2=[2,4,6]; num3=[7,8,9];  
nums=num1.concat(num2,num3) // creates array [1,3,5,2,4,6,7,8,9]
```

[CONTENTS](#) 

String.concat

Syntax

```
myString.concat(value1,...valueN);
```

Arguments value1,...valueN Zero or more values to be concatenated.

Description Method; combines the specified values and returns a new string. If necessary, each `value` argument is converted to a string and appended, in order, to the end of the string.

Player Flash 5 or later.

[CONTENTS](#) 

XMLSocket.connect

Syntax

```
myXMLSocket.connect(host, port);
```

Arguments **host** A fully qualified DNS domain name, or a IP address in the form aaa.bbb.ccc.ddd. You can also specify null to connect to the host server on which the movie resides.

port The TCP port number on the host used to establish a connection. The port number must be 1024 or higher.

Description Method; establishes a connection to the specified Internet host using the specified TCP port (must be 1024 or higher), and returns true or false depending on whether a connection is successfully established. If you don't know the port number of your Internet host machine, contact your network administrator. If the Flash Netscape plug-in or ActiveX control is being used, the host specified in the argument must have the same subdomain as the host from where the movie was downloaded.

If you specify null for the **host** argument, the host contacted will be the host where the movie calling XMLSocket.connect resides. For example, if the movie was downloaded from http://www.yoursite.com, specifying null for the host argument is the same as entering the IP address for www.yoursite.com.

If XMLSocket.connect returns a value of true, the initial stage of the connection process is successful; later, the XMLSocket.onConnect method is invoked to determine whether the final connection succeeded or failed. If XMLSocket.connect returns false, a connection could not be established.

Player Flash 5 or later.

Example The following example uses XMLSocket.connect to connect to the host where the movie resides, and uses trace to display the return value indicating the success or failure of the connection:

```
function myOnConnect(success) {  
    if (success) {  
        trace ("Connection succeeded!")  
    } else {  
        trace ("Connection failed!")  
    }  
}  
socket = new XMLSocket()  
socket.onConnect = myOnConnect  
if (!socket.connect(null, 2000)) {  
    trace ("Connection failed!")  
}
```

See also

[function
XMLSocket.onConnect](#)

[CONTENTS](#) 

Date (object)

The Date object allows you to retrieve date and time values relative to universal time (Greenwich Mean Time, now called Universal Coordinated Time) or relative to the operating system on which the Flash Player is running. To call the methods of the Date object, you must first create an instance of the Date object using the constructor.

The Date object requires the Flash 5 Player.

The methods of the Date object are not static, but apply only to the individual instance of the Date object specified when the method is called.

Method summary for Date object

getDate	Returns the day of the month of the specified Date object according to local time.
getDay	Returns the day of the month for the specified Date object according to local time.
getFullYear	Returns the four-digit year of the specified Date object according to local time.
getHours	Returns the hour of the specified Date object according to local time.
getMilliseconds	Returns the milliseconds of the specified Date object according to local time.
getMinutes	Returns the minutes of the specified Date object according to local time.
getMonth	Returns the month of the specified Date object according to local time.
getSeconds	Returns the seconds of the specified Date object according to local time.
getTime	Returns the number of milliseconds since midnight January 1, 1970, universal time, for the specified Date object.
getTimezoneOffset	Returns the difference, in minutes, between the computer's local time and the universal time.
getUTCDate	Returns the day (date) of the month of the specified Date object according to universal time.
getUTCDay	Returns the day of the week of the specified Date object according to universal time.
getUTCFullYear	Returns the four-digit year of the specified Date object according to universal time.
getUTCHours	Returns the hour of the specified Date object according to universal time.
getUTCMilliseconds	Returns the milliseconds of the specified Date object according to universal time.
getUTCMinutes	Returns the minute of the specified Date object according to universal time.
getUTCMonth	Returns the month of the specified Date object according to universal time.
getUTCSeconds	Returns the seconds of the specified Date object according to universal time.
getYear	Returns the year of the specified Date object according to local time.
setDate	Returns the day of the month of a specified Date object according to local time.
setFullYear	Sets the full year for a Date object according to local time.
setHours	Sets the hours for a Date object according to local time.
setMilliseconds	Sets the milliseconds for a Date object according to local time.

<code>setMinutes</code>	Sets the minutes for a Date object according to local time.
<code>setMonth</code>	Sets the month for a Date object according to local time.
<code>setSeconds</code>	Sets the seconds for a Date object according to local time.
<code>setTime</code>	Sets the date for the specified Date object in milliseconds.
<code>setUTCDate</code>	Sets the date of the specified Date object according to universal time.
<code>setUTCFullYear</code>	Sets the year of the specified Date object according to universal time.
<code>setUTCHours</code>	Sets the hour of the specified Date object according to universal time.
<code>setUTCMilliseconds</code>	Sets the milliseconds of the specified Date object according to universal time.
<code>setUTCMinutes</code>	Sets the minute of the specified Date object according to universal time.
<code>setUTCMonth</code>	Sets the month represented by the specified Date object according to universal time.
<code>setUTCSeconds</code>	Sets the seconds of the specified Date object according to universal time.
<code>setYear</code>	Sets the year for the specified Date object according to local time.
<code>toString</code>	Returns a string value representing the date and time stored in the specified Date object.
<code>Date.UTC</code>	Returns the number of milliseconds between midnight on January 1, 1970, universal time, and the specified time.

Constructor for the Date object

Syntax

```
new Date( );
new Date(year [, month [, date [, hour [, minute [, second [, millisecond ]]]]] ] ) ;
```

Arguments year A value of 0 to 99 indicates 1900 though 1999, otherwise all 4 digits of the year must be specified.

month An integer from 0 (January) to 11 (December). This argument is optional.

date An integer from 1 to 31. This argument is optional.

hour An integer from 0 (midnight) to 23 (11 p.m.).

minute An integer from 0 to 59. This argument is optional.

second An integer from 0 to 59. This argument is optional.

millisecond An integer from 0 to 999. This argument is optional.

Description Object; constructs a new Date object holding the current date and time.

Player Flash 5 or later.

Example The following example retrieves the current date and time:

```
now = new Date( );
```

The following example creates a new Date object for a Gary's birthday, August 7, 1974:

```
gary_birthday = new Date (74, 7, 7);
```

The following example creates a new Date object, concatenates the returned values of the Date object methods `getMonth`, `getDate`, and `getFullYear`, and displays them in the text field specified by the variable `dateTextField`.

```
myDate = new Date();
dateTextField = (mydate.getMonth() + "/" + myDate.getDate() + "/" + mydate.getFullYear());
```

[CONTENTS](#) 

Number (object)

The Number object is a simple wrapper object for the number data type, which means that you can manipulate primitive numeric values using the methods and properties associated with the Number object. The functionality provided by this object is identical to that of the JavaScript Number object.

You must use the Number constructor when calling the methods of the Number object, but you do not need to use the constructor when calling the properties of the Number object. The following examples specify the syntax for calling the methods and properties of the Number object:

This is an example of calling the toString method of the Number object:

```
myNumber = new Number(1234);  
myNumber.toString();
```

Returns a string containing the binary representation of the number 1234.

This is an example of calling the MIN_VALUE property (also called a constant) of the Number object:

```
smallest = Number.MIN_VALUE
```

Method summary for the Number object

toString	Returns the string representation of a Number object.
valueOf	Returns the primitive value of a Number object.

Property summary for the Number object

MAX_VALUE	Constant representing the largest representable number (double-precision IEEE-754). This number is approximately 1.7976931348623158e+308.
MIN_VALUE	Constant representing the smallest representable number (double-precision IEEE-754). This number is approximately 5e-324.
NaN	Constant representing the value for Not a Number (NaN).
NEGATIVE_INFINITY	Constant representing the value for negative infinity.
POSITIVE_INFINITY	Constant representing the value for positive infinity. This value is the same as the global variable Infinity.

Constructor for the Number object

Syntax

```
myNumber = new Number(value);
```

Arguments value The numeric value of the Number object being created, or a value to be converted to a number.

Description Constructor; creates a new Number object. You must use the Number constructor when using the toString and valueOf methods of the Number object. You do not use a constructor when using the properties of the Number object. The new Number constructor is primarily used as a placeholder. An instance of the Number object is

not the same as the `Number` function that converts an argument to a primitive value.

Player Flash 5 or later.

Example The following code constructs new `Number` objects:

```
n1 = new Number(3.4); n2 = new Number(-10);
```

See also

[Number \(function\)](#)

[CONTENTS](#) 

©1995-2001 Macromedia, Inc. All rights reserved.

[Privacy policy](#) | [Contact us](#) | [Feedback](#)

Object (object)

The generic Object object is at the root of the ActionScript class hierarchy. The functionality of the generic Object object is a small subset of that provided by the JavaScript Object object.

The generic Object object requires the Flash 5 Player.

Method summary for the Object object

toString	Converts the specified object to a string, and returns it.
valueOf	Returns the primitive value of an Object object.

Constructor for the Object object

Syntax

```
new Object();  
new Object(value);
```

Arguments `value` A number, Boolean, or string to be converted to an object. This argument is optional. If you do not specify `value`, the constructor creates a new object with no defined properties.

Description Constructor; creates a new Object object.

Player Flash 5 or later.

See also

[Sound.setTransform](#) [Color.setTransform](#)

CONTENTS 

Sound (object)

The Sound object allows you to set and control sounds in a particular movie clip instance, or for the global Timeline, if you do not specify a `target` when creating a new sound object. You must use the constructor `new Sound` to create an instance of the Sound object before calling the methods of the Sound object.

The Sound object is only supported for the Flash 5 Player.

Method summary for the Sound object

<code>attachSound</code>	Attaches the sound specified in the argument.
<code>getPan</code>	Returns the value of the previous <code>setPan</code> call.
<code>getTransform</code>	Returns the value of the previous <code>setTransform</code> call.
<code>getVolume</code>	Returns the value of the previous <code>setVolume</code> call.
<code>setPan</code>	Sets the left/right balance of the sound.
<code>setTransform</code>	Sets transform for a sound.
<code>setVolume</code>	Sets the volume level for a sound.
<code>start</code>	Starts playing a sound from the beginning or, optionally, from an offset point set in the argument.
<code>stop</code>	Stops the specified sound or all sounds currently playing.

Constructor for the Sound object

Syntax

```
new Sound();  
new Sound(target);
```

Arguments `target` The movie clip instance that the Sound object applies to. This argument is optional.

Description Method; creates a new Sound object for a specified movie clip. If you do not specify a `target`, the Sound object controls all of the sounds in the global Timeline.

Player Flash 5 or later.

Example

```
GlobalSound = new Sound(); MovieSound = new Sound(mymovie);
```

[CONTENTS](#) 

String (object)

The String object is a wrapper for the string primitive data type, which allows you to use the methods and properties of the String object to manipulate primitive string value types. You can convert the value of any object into a string using the `String()` function.

All of the methods of the String object, except for `concat`, `fromCharCode`, `slice`, and `substr`, are generic. This means the methods themselves call `this.toString` before performing their operations, and you can use these methods with other non-String objects.

You can call any of the methods of the String object using the constructor method `new String` or using a string literal value. If you specify a string literal, the ActionScript interpreter automatically converts it to a temporary String object, calls the method, and then discards the temporary String object. You can also use the `String.length` property with a string literal.

It is important that you do not confuse a string literal with an instance of the String object. In the following example the first line of code creates the string literal `s1`, and the second line of code creates an instance of the String object `s2`.

```
s1 = "foo"
s2 = new String("foo")
```

It is recommended that you use string literals unless you specifically need to use a String object, as String objects can have counterintuitive behavior.

Method summary for String object

<code>charAt</code>	Returns a number corresponding to the placement of the character in the string.
<code>charCodeAt</code>	Returns the value of the character at the given index as a 16-bit integer between 0 and 65535.
<code>concat</code>	Combines the text of two strings and returns a new string.
<code>fromCharCode</code>	Returns a string made up of the characters specified in the arguments.
<code>indexOf</code>	Searches the string and returns the index of the value specified in the arguments. If value occurs more than once, the index of the first occurrence is returned. If value is not found, -1 is returned.
<code>lastIndexOf</code>	Returns the last occurrence of substring within the string that appears before the start position specified in the argument, or -1 if not found.
<code>slice</code>	Extracts a section of a string and returns a new string.
<code>split</code>	Splits a String object into an array of strings by separating the string into substrings.
<code>substr</code>	Returns a specified number of characters in a string, beginning at the location specified in the argument.
<code>substring</code>	Returns the characters between two indexes, specified in the arguments, into the string.
<code>toLowerCase</code>	Converts the string to lowercase and returns the result.
<code>toUpperCase</code>	Converts the string to uppercase and returns the result.

Property summary for the String object

length	Returns the length of the string.

Constructor for the String object

Syntax

new String(value);

Arguments value The initial value of the new String object.

Description Constructor; creates a new String object.

Player Flash 5 or later.

See also

[String \(function\)](#)
[" " \(string delimiter\)](#)

CONTENTS 

XML (object)

Use the methods and properties of the XML object to load, parse, send, build, and manipulate XML document trees.

You must use the constructor `new XML ()` to create an instance of the XML object before calling any of the methods of the XML object.

XML is supported by Flash 5 or later versions of the Flash Player.

Method summary for the XML object

<code>appendChild</code>	Appends a node to the end of the specified object's child list.
<code>cloneNode</code>	Clones the specified node and, optionally, recursively clones all children.
<code>createElement</code>	Creates a new XML element.
<code>createTextNode</code>	Creates a new XML text node.
<code>hasChildNodes</code>	Returns <code>true</code> if the specified node has child nodes; otherwise, returns <code>false</code> .
<code>insertBefore</code>	Inserts a node in front of an existing node in the specified node's child list.
<code>load</code>	Loads a document (specified by the XML object) from a URL.
<code>onLoad</code>	A callback function for load and sendAndLoad.
<code>parseXML</code>	Parses an XML document into the specified XML object tree.
<code>removeNode</code>	Removes the specified node from its parent.
<code>send</code>	Sends the specified XML object to a URL.
<code>sendAndLoad</code>	Sends the specified XML object to a URL and loads the server response into another XML object.
<code>toString</code>	Converts the specified node and any children to XML text.

Property summary for the XML object

<code>doctypeDecl</code>	Sets and returns information about an XML document's DOCTYPE declaration.
<code>firstChild</code>	References the first child in the list for the specified node.
<code>lastChild</code>	References the last child in the list for the specified node.
<code>loaded</code>	Checks if the specified XML object has loaded.
<code>nextSibling</code>	References the next sibling in the parent node's child list.
<code>nodeName</code>	Returns the tag name of an XML element.
<code>nodeType</code>	Returns the type of the specified node (XML element or text node).

nodeValue	Returns the text of the specified node if the node is a text node.
parentNode	References the parent node of the specified node.
previousSibling	References the previous sibling in the parent node's child list.
status	Returns a numeric status code indicating the success or failure of an XML document parsing operation.
xmlDecl	Sets and returns information about an XML document's document declaration.

Collections summary for the XML object

attributes	Returns an associative array containing all of the attributes of the specified node.
childNodes	Returns an array containing references to the child nodes of the specified node.

Constructor for the XML object

Syntax

```
new XML( ) ;
new XML(source);
```

Arguments `source` The XML document parsed to create the new XML object.

Description Constructor; creates a new XML object. You must use the constructor method to create an instance of the XML object before calling any of the XML object methods.

The first syntax constructs a new, empty XML object.

The second syntax constructs a new XML object by parsing the XML document specified in the `source` argument, and populates the newly created XML object with the resulting XML document tree.

Note: The `createElement` and `createTextNode` methods are the 'constructor' methods for creating the elements and text nodes in an XML document tree.

Player Flash 5 or later.

Example The following example creates a new empty XML object:

```
myXML = new XML( ) ;
```

See also

[XML.createTextNode](#)
[XML.createElement](#)

CONTENTS 

XMLSocket (object)

The XMLSocket object implements client sockets that allow the computer running the Flash Player to communicate with a server computer identified by an IP address or domain name.

Using the XMLSocket object

To use the XMLSocket object, the server computer must run a daemon that understands the protocol used by the XMLSocket object. The protocol is as follows:

- ▶ XML messages are sent over a full-duplex TCP/IP stream socket connection.
- ▶ Each XML message is a complete XML document, terminated by a zero byte.
- ▶ An unlimited number of XML messages can be sent and received over a single XMLSocket connection.

The XMLSocket object is useful for client-server applications that require low latency, such as real-time chat systems. A traditional HTTP-based chat solution frequently polls the server and downloads new messages using an HTTP request. In contrast, an XMLSocket chat solution maintains an open connection to the server, which allows the server to immediately send incoming messages without a request from the client.

Setting up a server to communicate with the XMLSocket object can be challenging. If your application does not require real-time interactivity, use the `loadVariables` action, or Flash's HTTP-based XML server connectivity (`XML.load`, `XML.sendAndLoad`, `XML.send`), instead of the XMLSocket object.

To use the methods of the XMLSocket object, you must first use the constructor, `new XMLSocket`, to create a new XMLSocket object.

XMLSocket and security

Because the XMLSocket object establishes and maintains an open connection to the server, the following restrictions have been placed on the XMLSocket object for security reasons:

- ▶ The `XMLSocket.connect` method can connect only to TCP port numbers greater than or equal to 1024. One consequence of this restriction is that the server daemons that communicate with the XMLSocket object must also be assigned to port numbers greater than or equal to 1024. Port numbers below 1024 are often used by system services such as FTP, Telnet, and HTTP, thus barring the XMLSocket object from these ports. The port number restriction limits the possibility that these resources will be inappropriately accessed and abused.
- ▶ The `XMLSocket.connect` method can connect only to computers in the same subdomain where the SWF file (movie) resides. This restriction does not apply to movies running off a local disk. (This restriction is identical to the security rules for `loadVariables`, `XML.sendAndLoad`, and `XML.load`.)

Method summary for the XMLSocket object

<code>close</code>	Closes an open socket connection.
<code>connect</code>	Establishes a connection to the specified server.
<code>onClose</code>	A callback function that is invoked when an XMLSocket connection is closed.
<code>onConnect</code>	A callback function that is invoked when an XMLSocket connection is established.
<code>onXML</code>	A callback function that is invoked when an XML object arrives from the server.
<code>send</code>	Sends an XML object to the server.

Constructor for the XMLSocket object

Syntax

```
new XMLSocket( );
```

Arguments None.

Description Constructor; creates a new XMLSocket object. The XMLSocket object is not initially connected to any server. You must call the XMLSocket.connect method to connect the object to a server.

Player Flash 5 or later.

Example

```
myXMLSocket = new XMLSocket( );
```

See also

[XMLSocket.connect](#)

[CONTENTS](#) 

Key.CONTROL

Syntax

Key . CONTROL

Arguments None.

Description Property; constant associated with the key code value for the Control key (17).

Player Flash 5 or later.

[CONTENTS](#) 

Math.cos

Syntax

`Math.cos(x) ;`

Arguments x An angle measured in radians.

Description Method; returns the cosine (a value from -1.0 to 1.0) of the angle specified by the argument x. The angle x must be specified in radians. Use the information outlined in the introduction to the Math object to calculate a radian.

Player Flash 5 or later. In the Flash 4 Player, the methods and properties of the Math object are emulated using approximations and may not be as accurate as the non-emulated math functions supported by the Flash 5 Player.

[CONTENTS](#) 

XML.createElement

Syntax `myXML.createElement (name) ;`

Arguments `name` The tag name of the XML element being created.

Description Method; creates a new XML element with the name specified in the argument. The new element initially has no parent and no children. The method returns a reference to the newly created XML object representing the element. This method and `createTextNode` are the constructor methods for creating nodes for an XML object.

Player Flash 5 or later.

[CONTENTS](#) 

XML.createTextNode

Syntax `myXML.createTextNode (text) ;`

Arguments `text` The text used to create the new text node.

Description Method; creates a new XML text node with the specified text. The new node initially has no parent, and text nodes cannot have children. This method returns a reference to the XML object representing the new text node. This method and `createElement` are the constructor methods for creating nodes for an XML object.

Player Flash 5 or later.

[CONTENTS](#) 

Key.DELETEKEY

Syntax

Key.DELETEKEY

Arguments None.

Description Property; constant associated with the key code value for the Delete key (46).

Player Flash 5 or later.

[CONTENTS](#) 

XML.docTypeDecl

Syntax `myXML.XMLdocTypeDecl ;`

Arguments None.

Description Property; sets and returns information about the XML document DOCTYPE declaration. After the XML text has been parsed into an XML object, the `XML.docTypeDecl` property of the XML object is set to the text of the XML document's DOCTYPE declaration. For example, `<!DOCTYPE greeting SYSTEM "hello.dtd">`. This property is set using a string representation of the DOCTYPE declaration, not an XML node object.

ActionScript's XML parser is not a validating parser. The DOCTYPE declaration is read by the parser and stored in the `docTypeDecl` property, but no DTD validation is performed.

If no DOCTYPE declaration was encountered during a parse operation, `XML.docTypeDecl` is set to undefined. `XML.toString` outputs the contents of `XML.docTypeDecl` immediately after the XML declaration stored in `XML.xmlDecl`, and before any other text in the XML object. If `XML.docTypeDecl` is undefined, no DOCTYPE declaration is output.

Player Flash 5 or later.

Example The following example uses `XML.docTypeDecl` to set the DOCTYPE declaration for an XML object.

```
myXML.docTypeDecl = "<!DOCTYPE greeting SYSTEM \"hello.dtd\">";
```

See also

[XML.toString XML.xmlDecl](#)

[CONTENTS](#) 

Key.DOWN

x20941 | IDS_ACTIONHELP_KEY_DOWN2, Key.DOWN

Syntax

Key.DOWN

Arguments None.

Description Property; constant associated with the key code value for the Down Arrow key (40).

Player Flash 5 or later.

[CONTENTS](#) 

MovieClip.duplicateMovieClip

x208C9 | IDS_ACTIONHELP_CLIP_DUPLICATEMOVIECLIP, MovieClip.duplicateMovieClip

Syntax

```
anyMovieClip.duplicateMovieClip(newname, depth);
```

Arguments `newname` A unique identifier for the duplicate movie clip.

`depth` A number specifying the depth level where the movie specified is to be placed.

Description Method; creates an instance of the specified movie clip while the movie is playing. Duplicated movie clips always start playing at frame 1, no matter what frame the original movie clip is on when the `duplicateMovieClip` method is called. Variables in the parent movie clip are not copied into the duplicate movie clip. If the parent movie clip is deleted the duplicate movie clip is also deleted. Movie clips added with `duplicateMovieClip` can be deleted with `removeMovieClip` action or method.

Player Flash 5 or later.

See also

[removeMovieClip](#) [MovieClip.removeMovieClip](#)

CONTENTS 

Math.E

Syntax

Math.E

Arguments None.

Description Constant; a mathematical constant for the base of natural logarithms, expressed as e. The approximate value of e is 2.71828.

Player Flash 5 or later. In the Flash 4 Player, the methods and properties of the Math object are emulated using approximations and may not be as accurate as the non-emulated math functions supported by the Flash 5 Player.

[CONTENTS](#) 

Key.END

x20942 | IDS_ACTIONHELP_KEY_END2, Key.END

Syntax

Key . END

Arguments None.

Description Property; constant associated with the key code value for the End key (35).

Player Flash 5 or later.

[CONTENTS](#) 

Key.ENTER

x20943 | IDS_ACTIONHELP_KEY_ENTER2, Key.ENTER

Syntax

Key.ENTER

Arguments None.

Description Property; constant associated with the key code value for the Enter key (13).

Player Flash 5 or later.

[CONTENTS](#) 

Key.ESCAPE

x20944 | IDS_ACTIONHELP_KEY_ESCAPE2, Key.ESCAPE

Syntax

Key.ESCAPE

Arguments None.

Description Property; constant associated with the key code value for the Escape key (27).

Player Flash 5 or later.

[CONTENTS](#) 

Math.exp

Syntax

```
Math.exp(x) ;
```

Arguments x The exponent; a number or expression.

Description Method; returns the value of the base of the natural logarithm (e), to the power of the exponent specified in the argument x. The constant Math.E can provide the value of e.

Player Flash 5 or later. In the Flash 4 Player, the methods and properties of the Math object are emulated using approximations and may not be as accurate as the non-emulated math functions supported by the Flash 5 Player.

[CONTENTS](#) 

XML.firstChild

Syntax

```
myXML.firstChild;
```

Arguments None.

Description Property (read-only); evaluates the specified XML object and references the first child in the parent node's children list. This property is null if the node does not have children. This property is undefined if the node is a text node. This is a read-only property and cannot be used to manipulate child nodes; use the methods `appendChild`, `insertBefore`, and `removeNode` to manipulate child nodes.

Player Flash 5 or later.

See also

[XML.appendChild](#) [XML.insertBefore](#) [XML.removeNode](#)

CONTENTS 

Math.floor

Syntax

```
Math.floor(x);
```

Arguments x A number or expression.

Description Method; returns the floor of the number or expression specified in the argument x. The floor is the closest integer that is less than or equal to the specified number or expression.

Player Flash 5 or later. In the Flash 4 Player, the methods and properties of the Math object are emulated using approximations and may not be as accurate as the non-emulated math functions supported by the Flash 5 Player.

Example The following returns a value of 12:

```
Math.floor(12.5);
```

[CONTENTS](#) 

String.fromCharCode

Syntax

```
myString.fromCharCode(c1,c2,...cN);
```

Arguments c1,c2,...cN The characters to be made into a string.

Description Method; returns a string made up of the characters specified in the arguments.

Player Flash 5 or later.

[CONTENTS](#) 

Key.getAscii

Syntax

```
Key.getAscii();
```

Arguments None.

Description Method; returns the ASCII code of the last key pressed or released.

Player Flash 5 or later.

[CONTENTS](#) 

Selection.getBeginIndex

Syntax

```
Selection.getBeginIndex( ) ;
```

Arguments None.

Description Method; returns index at the beginning of the selection span. If no index exists or no field currently has the focus, the method returns -1. Selection span indexes are zero-based (where the first position is 0, the second position is 1, and so on).

Player Flash 5 or later.

[CONTENTS](#) 

MovieClip.getBounds

Syntax

```
anyMovieClip.getBounds(targetCoordinateSpace);
```

Arguments `targetCoordinateSpace` The target path of the Timeline whose coordinate space you want to use as a reference point.

Description Method; returns the minimum and maximum x and y coordinate values of the MovieClip for the target coordinate space specified in the argument. The return object will contain the properties {xMin, xMax, yMin, yMax}. Use the `localToGlobal` and `globalToLocal` methods of the MovieClip object to convert the movie clip's local coordinates to Stage coordinates, or Stage coordinates to local coordinates respectively.

Player Flash 5 or later.

Example The following example uses `getBounds` to retrieve the bounding box of the `myMovieClip` instance in the coordinate space of the main movie:

```
myMovieClip.getBounds(._root);
```

See also

[MovieClip.globalToLocal](#) [MovieClip.localToGlobal](#)

CONTENTS 

MovieClip.getBytesLoaded

Syntax

```
anyMovieClip.getBytesLoaded();
```

Arguments None.

Description Method; returns the number of bytes loaded (streamed) for the specified Movie Clip object. Because internal movie clips load automatically, the return result for this method and MovieClip.getBytesTotal will be the same if the specified Movie Clip object references an internal movie clip. This method is intended for use on loaded movies. You can compare the value of getBytesLoaded with the value of getBytesTotal to determine what percentage of an external movie has loaded.

Player Flash 5 or later.

[CONTENTS](#) 

MovieClip.getBytesTotal

Syntax

```
anyMovieClip.getBytesTotal();
```

Arguments None.

Description Method; returns the size, in bytes, of the specified Movie Clip object. For movie clips that are external, (the root movie or a movie clip that is being loaded into a target or a level) the return value is the size of the SWF file.

Player Flash 5 or later.

[CONTENTS](#) 

Selection.getCaretIndex

Syntax

```
Selection.getCaretIndex( ) ;
```

Arguments None.

Description Method; returns the index of the blinking cursor position. If there is no blinking mouse pointer displayed, the method returns -1. Selection span indexes are zero-based (where the first position is 0, the second position is 1, and so on).

Player Flash 5 or later.

[CONTENTS](#) 

Key.getCode

Syntax

```
Key.getCode( ) ;
```

Arguments None.

Description Method; returns the key code value of the last key pressed. Use the information in Appendix B, "Keyboard Keys and Key Code Values," to match the returned key code value with the virtual key on a standard keyboard.

Player Flash 5 or later.

[CONTENTS](#) 

Date.getDate

Syntax

```
myDate.getDate( ) ;
```

Arguments None.

Description Method; returns the day of the month (an integer from 1 to 31) of the specified Date object according to local time.

Player Flash 5 or later.

[CONTENTS](#) 

Date.getDay

Syntax

```
myDate.getDay( ) ;
```

Arguments None.

Description Method; returns the day of the month (0 for Sunday, 1 for Monday, and so on) of the specified Date object according to local time. Local time is determined by the operating system on which the Flash Player is running.

Player Flash 5 or later.

[CONTENTS](#) 

Selection.getEndIndex

Syntax

```
Selection.getEndIndex( ) ;
```

Arguments None.

Description Method; returns the ending index of the currently focused selection span. If no index exists, or if there is no currently focused selection span, the method returns -1. Selection span indexes are zero-based (where the first position is 0, the second position is 1, and so on).

Player Flash 5 or later.

[CONTENTS](#) 

Selection.setFocus

Syntax

```
Selection.setFocus( );
```

Arguments None.

Description Method; returns the name of the variable of the currently focused editable text field. If no text field is currently focused, the method returns null.

Player Flash 5 or later.

Example The following code returns the name of the variable:

```
_root.anyMovieClip.myTextField.
```

[CONTENTS](#) 

Date.getFullYear

Syntax

```
myDate.getFullYear();
```

Arguments None.

Description Method; returns the full year (a four-digit number, for example, 2000) of the specified Date object, according to local time. Local time is determined by the operating system on which the Flash Player is running.

Player Flash 5 or later.

Example The following example uses the constructor to create a new Date object and send the value returned by the getFullYear method to the Output window:

```
myDate = new Date();  
trace(myDate.getFullYear());
```

[CONTENTS](#) 

Date.getHours

Syntax

```
myDate.getHours( ) ;
```

Arguments None.

Description Method; returns the hour (an integer from 0 to 23) of the specified Date object, according to local time. Local time is determined by the operating system on which the Flash Player is running.

Player Flash 5 or later.

[CONTENTS](#) 

Date.getMilliseconds

Syntax

```
myDate.getMilliseconds( );
```

Arguments None.

Description Method; returns the milliseconds (an integer from 0 to 999) of the specified Date object, according to local time. Local time is determined by the operating system on which the Flash Player is running.

Player Flash 5 or later.

[CONTENTS](#) 

Date.getMinutes

Syntax

```
myDate.getMinutes( );
```

Arguments None.

Description Method; returns the minutes (an integer from 0 to 59) of the specified Date object, according to local time. Local time is determined by the operating system on which the Flash Player is running.

Player Flash 5 or later.

[CONTENTS](#) 

Date.getMonth

Syntax

```
myDate.getMonth( ) ;
```

Arguments None.

Description Method; returns the month (0 for January, 1 for February, and so on) of the specified Date object, according to local time. Local time is determined by the operating system on which the Flash Player is running.

Player Flash 5 or later.

[CONTENTS](#) 

Sound.getPan

Syntax

```
mySound.getPan( ) ;
```

Arguments None.

Description Method; returns the pan level set in the last setPan call as an integer from -100 to 100. The pan setting controls the left-right balance of the current and future sounds in a movie.

This method is cumulative with the setVolume or setTransform methods.

Player Flash 5 or later.

See also [Sound.setPan](#)

[Sound.setTransform](#)

[CONTENTS](#) 

Color.setRGB

Syntax

```
myColor.setRGB(0xRRGGBB);
```

Arguments 0xRRGGBB The hexadecimal or RGB color to be set. RR, GG, and BB each consist of two hexadecimal digits specifying the offset of each color component.

Description Method; specifies an RGB color for the Color object. Calling this method overrides any previous settings by the setTransform method.

Player Flash 5 or later.

Example The following example sets the RGB color value for the movie clip myMovie:

```
myColor = new Color(myMovie); myColor.setRGB(0x993366);
```

See also

[Color.setTransform](#)

CONTENTS 

Date.getSeconds

Syntax

```
myDate.getSeconds ( ) ;
```

Arguments None.

Description Method; returns the seconds (an integer from 0 to 59) of the specified Date object, according to local time. Local time is determined by the operating system on which the Flash Player is running.

Player Flash 5 or later.

[CONTENTS](#) 

Date.getTime

Syntax

```
myDate.getTime();
```

Arguments None.

Description Method; returns the number of milliseconds (an integer from 0 to 999) since midnight January 1, 1970, universal time, for the specified Date object. Use this method to represent a specific instant in time when comparing two or more times defined in different time zones.

Player Flash 5 or later.

[CONTENTS](#) 

Date.getTimezoneOffset

Syntax

```
mydate.getTimezoneOffset();
```

Arguments None.

Description Method; returns the difference, in minutes, between the computer's local time and the universal time.

Player Flash 5 or later.

Example The following example returns the difference between the local daylight-saving time for San Francisco and the universal time. Daylight-savings time is factored into the returned result only if the date defined in the Date object is during the daylight-savings time.

```
new Date().getTimezoneOffset();
```

The result is as follows:

420 (7 hours * 60 minutes/hour = 420 minutes)

[CONTENTS](#) 

Color.getTransform

Syntax

```
myColor.getTransform( );
```

Arguments None.

Description Method; returns the transform value set by the last setTransform call.

Player Flash 5 or later.

See also

[Color.setTransform](#)

CONTENTS 

Sound.getTransform

Syntax

```
mySound.getTransform() ;
```

Arguments None.

Description Method; returns the sound transform information for the specified Sound object set with the last setTransform call.

Player Flash 5 or later.

See also [Sound.setTransform](#)

[CONTENTS](#) 

MovieClip.getURL

Syntax

```
anyMovieClip.getURL(URL [,window, variables]);
```

Arguments URL The URL from which to obtain the document.

window An optional argument specifying the name, frame, or expression specifying the window or HTML frame that the document is loaded into. You can also use one of the following reserved target names: `_self` specifies the current frame in the current window, `_blank` specifies a new window, `_parent` specifies the parent of the current frame, `_top` specifies the top-level frame in the current window.

variables An optional argument specifying a method for sending variables associated with the movie to load. If there are no variables, omit this argument; otherwise, specify whether to load variables using a GET or POST method. GET appends the variables to the end of the URL, and is used for small numbers of variables. POST sends the variables in a separate HTTP header and is used for long strings of variables.

Description Method; loads a document from the specified URL into the specified window. The `getURL` method can also be used to pass variables to another application defined at the URL using a GET or POST method.

Player Flash 5 or later.

[CONTENTS](#) 

Date.getUTCDate

Syntax

```
myDate.getUTCDate( ) ;
```

Arguments None.

Description Method; returns the day (date) of the month in the specified Date object, according to universal time.

Player Flash 5 or later.

[CONTENTS](#) 

Date.getUTCDay

Syntax

```
myDate.getUTCDate( ) ;
```

Arguments None.

Description Method; returns the day of the week of the specified Date object, according to universal time.

[CONTENTS](#) 

Date.getUTCFullYear

Syntax

```
myDate.getUTCFullYear( );
```

Arguments None.

Description Method; returns the four-digit year of the specified Date object, according to universal time.

Player Flash 5 or later.

[CONTENTS](#) 

Date.getUTCHours

Syntax

```
myDate.getUTCHours( ) ;
```

Arguments None.

Description Method; returns the hours of the specified Date object, according to universal time.

Player Flash 5 or later.

[CONTENTS](#) 

Date.getUTCMilliseconds

Syntax

```
myDate.getUTCMilliseconds( ) ;
```

Arguments None.

Description Method; returns the milliseconds of the specified Date object, according to universal time.

Player Flash 5 or later.

[CONTENTS](#) 

Date.getUTCMinutes

Syntax

```
myDate.getUTCMinutes( ) ;
```

Arguments None.

Description Method; returns the minutes of the specified Date object, according to universal time.

Player Flash 5 or later.

[CONTENTS](#) 

Date.getUTCMonth

Syntax

```
myDate.getUTCMonth( ) ;
```

Arguments None.

Description Method; returns the month of the specified Date object, according to universal time.

Player Flash 5 or later.

[CONTENTS](#) 

Date.getUTCSeconds

Syntax

```
myDate.getUTCSeconds( ) ;
```

Arguments None.

Description Method; returns the seconds in the specified Date object, according to universal time.

Player Flash 5 or later.

[CONTENTS](#) 

Sound.getVolume

Syntax

```
mySound.getVolume() ;
```

Arguments None.

Description Method; returns the sound volume level as an integer from 0 to 100, where 0 is off and 100 is full volume. The default setting is 100.

Player Flash 5 or later.

See also

[Sound.setVolume](#)

[CONTENTS](#) 

Date.getYear

Syntax

```
myDate.getYear( ) ;
```

Arguments None.

Description Method; returns the year of the specified Date object, according to local time. Local time is determined by the operating system on which the Flash Player is running. The year is the full year minus 1900. For example, the year 2000 is represented as 100.

Player Flash 5 or later.

[CONTENTS](#) 

MovieClip.globalToLocal

Syntax

```
anyMovieClip.globalToLocal(point);
```

Arguments `point` The name or identifier of an object created with the generic Object object specifying the x and y coordinates as properties.

Description Method; converts the `point` object from Stage (global) coordinates to the movie clip's (local) coordinates.

Player Flash 5 or later.

Example The following example converts the global x and y coordinates of the `point` object to the local coordinates of the movie clip:

```
onClipEvent(mouseMove) { point = new object(); point.x =  
_root._xmouse; point.y = _root._ymouse; globalToLocal(point);  
_root.out = _xmouse + " === " + _ymouse; _root.out2 = point.x + "  
=== " + point.y; updateAfterEvent(); }
```

See also

[MovieClip.localToGlobal](#) [MovieClip.getBounds](#)

CONTENTS 

MovieClip.gotoAndPlay

Syntax

```
anyMovieClip.gotoAndPlay(frame);
```

Arguments frame The frame number to which the playhead will be sent.

Description Method; starts playing the movie at the specified frame.

Player Flash 5 or later.

[CONTENTS](#) 

MovieClip.gotoAndStop

Syntax

```
anyMovieClip.gotoAndStop(frame);
```

Arguments frame The frame number to which the playhead will be sent.

Description Method; stops the movie playing at the specified frame.

Player Flash 5 or later.

[CONTENTS](#) 

XML.hasChildNodes

Syntax

```
myXML.hasChildNodes();
```

Arguments None.

Description Method; evaluates the specified XML object and returns true if there are child nodes; otherwise, returns false.

Player Flash 5 or later.

Example The following example uses the information from the XML object in a user-defined function:

```
if (rootNode.hasChildNodes()) {  
    myfunc (rootNode.firstChild);  
}
```

[CONTENTS](#) 

Mouse.hide

Syntax

```
Mouse.hide();
```

Arguments None.

Description Method; hides the cursor in a movie. The cursor is visible by default.

Player Flash 5 or later.

Example The following code, attached to a movie clip on the main Timeline, hides the standard cursor and sets the x and y positions of the customCursor movie clip instance to the x and y mouse positions in the main Timeline:

```
onClipEvent(enterFrame){  
    Mouse.hide();  
    customCursorMC_x = _root._xmouse;  
    customCursorMC_y = _root._ymouse;  
}
```

See also

[_xmouse](#)

[_ymouse](#)

[Mouse.show](#)

[CONTENTS](#) 

MovieClip.hitTest

Syntax

```
anyMovieClip.hitTest(x, y, shapeFlag);  
anyMovieClip.hitTest(target);
```

Arguments **x** The **x** coordinate of the hit area on the Stage.

y The **y** coordinate of the hit area on the Stage.

The **x** and **y** coordinates are defined in the global coordinate space.

target The target path of the hit area that may intersect or overlap with the instance specified by **anyMovieClip**. The **target** usually represents a button or text-entry field.

shapeFlag A Boolean value specifying whether to evaluate the entire shape of the specified instance (**true**), or just the bounding box (**false**). This argument can only be specified if the hit area is identified using **x** and **y** coordinate arguments.

Description Method; evaluates the instance specified by **anyMovieClip** to see if it overlaps or intersects with the hit area identified by the **target** or **x** and **y** coordinate arguments.

Usage 1 compares the **x** and **y** coordinates to the shape or bounding box of the specified instance, according to the **shapeFlag** setting. If **shapeFlag** is set to **true**, only the area actually occupied by the instance on the Stage is evaluated, and if **x** and **y** overlap at any point, a value of **true** is returned. This is useful for determining if the movie clip is within a specified hit, or hotspot, area.

Usage 2 evaluates the bounding boxes of the **target** and specified instance, and returns **true** if they overlap or intersect at any point.

Player Flash 5 or later.

Example The following example uses **hitTest** with the **x_mouse** and **y_mouse** properties to determine whether the mouse is over the target's bounding box:

```
if (hitTest( _root._xmouse, _root._ymouse, false));
```

The following example uses **hitTest** to determine if the movie clip **ball** overlaps or intersects with the movie clip **square**:

```
if(_root.ball, hitTest(_root.square)){ trace("ball intersects  
square"); }
```

See also

[MovieClip.localToGlobal](#) [MovieClip.globalToLocal](#) [MovieClip.getBounds](#)

[CONTENTS](#) 

Key.HOME

Syntax

Key . HOME

Arguments None.

Description Property; constant associated with the key code value for the Home key (36).

Player Flash 5 or later.

[CONTENTS](#) 

String.indexOf

Syntax

```
myString.indexOf(value);  
myString.index of (value, start);
```

Arguments **value** An integer or string specifying the substring to be searched for within myString.

start An integer specifying the starting point of the substring. This argument is optional.

Description Method; searches the string and returns the position of the first occurrence of the specified **value**. If the **value** is not found, the method returns -1.

Player Flash 5 or later.

[CONTENTS](#) 

Key.INSERT

Syntax

Key . INSERT

Arguments None.

Description Property; constant associated with the key code value for the Insert key (45).

Player Flash 5 or later.

[CONTENTS](#) 

XML.insertBefore

Syntax

```
myXML.insertBefore(childNode, beforeNode);
```

Arguments childNode The node to be inserted.

beforeNode The node before the insertion point for the childNode.

Description Method; inserts a new child node into the XML object's child list, before the beforeNode.

Player Flash 5 or later.

[CONTENTS](#) 

Key.isDown

Syntax

```
Key.isDown(keycode);
```

Arguments `keycode` The key code value assigned to a specific key, or a Key object property associated with a specific key. Appendix B, "Keyboard Keys and Key Code Values," lists all of the key codes associated with the keys on a standard keyboard.

Description Method; returns true if the key specified in `keycode` is pressed. On the Macintosh, the key code values for the Caps Lock and Num Lock keys are identical.

Player Flash 5 or later.

[CONTENTS](#) 

Key.isToggled

Syntax

`Key.isToggled(keycode)`

Arguments `keycode` The key code for Caps Lock (20) or Num Lock (144).

Description Method; returns `true` if the Caps Lock or Num Lock key is activated (toggled). On the Macintosh, the key code values for these keys are identical.

Player Flash 5 or later.

[CONTENTS](#) 

Array.join

Syntax

```
myArray.join();  
myArray.join(separator);
```

Arguments separator A character or string that separates array elements in the returned string. If you omit this argument, a comma is used as the default separator.

Description Method; converts the elements in an array to strings, concatenates them, inserts the specified separator between the elements, and returns the resulting string.

Player Flash 5 or later.

Example The following example creates an array, with three elements. It then joins the array three times: using the default separator, then a comma and a space, and then a plus sign.

```
a = new Array("Earth","Moon","Sun") // assigns "Earth,Moon,Sun" to  
myVar1 myVar1=a.join(); // assigns "Earth, Moon, Sun" to myVar2  
myVar2=a.join(", "); // assigns "Earth + Moon + Sun" to myVar3  
myVar3=a.join(" + ");
```

[CONTENTS](#) 

Key (object)

The Key object is a top-level object that you can access without using a constructor. Use the methods for the Key object to build an interface that can be controlled by a user with a standard keyboard. The properties of the Key object are constants representing the keys most commonly used to control games. See Appendix B, "Keyboard Keys and Key Code Values," for a complete list of key code values.

Example

```
onClipEvent (enterFrame) {  
    if(Key.isDown(Key.RIGHT)) {  
        setProperty ("", _x, _x+10);  
    }  
}  
or  
onClipEvent (enterFrame) {  
    if(Key.isDown(39)) {  
        setProperty("", _x, _x+10);  
    }  
}
```

Method summary for the Key object

<code>getAscii;</code>	Returns the ASCII value of the last key pressed.
<code>getCode;</code>	Returns the virtual key code of the last key pressed.
<code>isDown;</code>	Returns <code>true</code> if the key specified in the argument is pressed.
<code>isToggled;</code>	Returns <code>true</code> if the Num Lock or Caps Lock key is activated.

Property summary for the Key object

All of the properties for the Key object are constants.

<code>BACKSPACE</code>	Constant associated with the key code value for the Backspace key (9).
<code>CAPSLock</code>	Constant associated with the key code value for the Caps Lock key (20).
<code>CONTROL</code>	Constant associated with the key code value for the Control key (17).
<code>DELETEKEY</code>	Constant associated with the key code value for the Delete key (46).
<code>DOWN</code>	Constant associated with the key code value for the Down Arrow key (40).
<code>END</code>	Constant associated with the key code value for the End key (35).
<code>ENTER</code>	Constant associated with the key code value for the Enter key (13).
<code>ESCAPE</code>	Constant associated with the key code value for the Escape key (27).
<code>HOME</code>	Constant associated with the key code value for the Home key (36).

INSERT	Constant associated with the key code value for the Insert key (45).
LEFT	Constant associated with the key code value for the Left Arrow key (37).
PGDN	Constant associated with the key code value for the Page Down key (34).
PGUP	Constant associated with the key code value for the Page Up key (33).
RIGHT	Constant associated with the key code value for the Right Arrow key (39).
SHIFT	Constant associated with the key code value for the Shift key (16).
SPACE	Constant associated with the key code value for the Spacebar (32).
TAB	Constant associated with the key code value for the Tab key (9).
UP	Constant associated with the key code value for the Up Arrow key (38).

XML.lastChild

Syntax

```
myXML.lastChild;
```

Arguments None.

Description Property (read-only); evaluates the XML object and references the last child in the parent node's child list. This method returns null if the node does not have children. This is a read-only property and cannot be used to manipulate child nodes; use the methods `appendChild`, `insertBefore`, and `removeNode` to manipulate child nodes.

Player Flash 5 or later.

See also

[XML.appendChild](#) [XML.insertBefore](#) [XML.removeNode](#)

[CONTENTS](#) 

Key.LEFT

x20949 | IDS_ACTIONHELP_KEY_LEFT2, Key.LEFT

Syntax

Key.LEFT

Arguments None.

Description Property; constant associated with the key code value for the Left Arrow key (37).

Player Flash 5 or later.

[CONTENTS](#) 

length

Syntax

```
length(expression);  
length(variable);
```

Arguments expression Any string.

variable The name of a variable.

Description String function; returns the length of the specified string or variable name.

Player Flash 4 or later. This function, along with all of the string functions, has been deprecated in Flash 5. It is recommended that you use the methods and length property of the String object to perform the same operations.

Example The following example returns the value of the string Hello:

```
length("Hello");
```

The result is 5.

See also

[" " \(string delimiter\)](#)
[String.length](#)

[CONTENTS](#) 

Array.length

x2089F | IDS_ACTIONHELP_ARRAY_LENGTH, Array.length

Syntax

```
myArray.length;
```

Arguments None.

Description Property; contains the length of the array. This property is automatically updated when new elements are added to the array. During assignment `myArray[index] = value`; if index is a number, and index+1 is a greater than the length property, the length property is updated to index + 1.

Player Flash 5 or later.

Example The following code explains how the length property is updated:

```
//initial length is 0
myArray = new Array();
//myArray.length is updated to 1
myArray[0] = 'a';
//myArray.length is updated to 2
myArray[1] = 'b';
//myArray.length is updated to 10
myArray[9] = 'c';
```

[CONTENTS](#) 

String.length

Syntax

`string.length`

Arguments None.

Description Property; returns the number of characters in the specified String object. The index of the last character for any string `x` is `x.length-1`.

Player Flash 5 or later.

[CONTENTS](#) 

Math.LN2

Syntax

Math.LN2

Arguments None.

Description Constant; a mathematical constant for the natural logarithm of 2, expressed as loge2, with an approximate value of 0.69314718055994528623.

Player Flash 5 or later. In the Flash 4 Player, the methods and properties of the Math object are emulated using approximations and may not be as accurate as the non-emulated math functions supported by the Flash 5 Player.

[CONTENTS](#) 

Math.LN10

Syntax

Math.LN10

Arguments None.

Description Constant; a mathematical constant for the natural logarithm of 10, expressed as $\log_e 10$, with an approximate value of 2.3025850929940459011.

Player Flash 5 or later. In the Flash 4 Player, the methods and properties of the Math object are emulated using approximations and may not be as accurate as the non-emulated math functions supported by the Flash 5 Player.

[CONTENTS](#) 

XML.load

Syntax

```
myXML.load(url);
```

Arguments `url` The URL where the XML document to be loaded is located. The URL must be in the same subdomain as the URL where the movie currently resides.

Description Method; loads an XML document from the specified URL, and replaces the contents of the specified XML object with the downloaded XML data. The load process is asynchronous; it does not finish immediately after the load method is executed. When load is executed, the XML object property `loaded` is set to `false`. When the XML data finishes downloading, the `loaded` property is set to `true`, and the `onLoad` method is invoked. The XML data is not parsed until it is completely downloaded. If the XML object previously contained any XML trees, they are discarded.

You can specify your own callback function in place of the `onLoad` method.

Player Flash 5 or later.

Example The following is a simple example using XML.load:

```
doc = new XML(); doc.load ( "theFile.xml" );
```

See also

[XML.onLoad](#)

[XML.loaded](#)

CONTENTS 

XML.loaded

Syntax

```
myXML.loaded;
```

Arguments None.

Description Property (read-only); determines whether the document loading process initiated by the XML.load call has completed. If the process completes successfully, the method returns true; otherwise, it returns false.

Player Flash 5 or later.

Example The following example uses XML.loaded in a simple script.

```
if (doc.loaded) {  
    gotoAndPlay(4)  
}
```

[CONTENTS](#) 

MovieClip.loadVariables

Syntax

```
anyMovieClip.loadVariables(url, variables);
```

Arguments `url` The absolute or relative URL for the external file. The host for the URL must be in the same subdomain as the movie clip.

`variables` The method for retrieving the variables. GET appends the variables to the end of the URL, and is used for small numbers of variables. POST sends the variables in a separate HTTP header and is used for long strings of variables.

Description Method; reads data from an external file and sets the values for variables in a movie or movie clip. The external file can be a text file generated by a CGI script, Active Server Pages (ASP), or PHP, and can contain any number of variables.

This method can also be used to update variables in the active movie with new values.

This method requires that the text at the URL be in the standard MIME format: application/x-www-urlformencoded (CGI script format).

Player Flash 5 or later.

See also

[MovieClip.loadMovie](#)

CONTENTS 

MovieClip.localToGlobal

Syntax

```
anyMovieClip.localToGlobal(point);
```

Arguments `point` The name or identifier of an object created with the `Object` object, specifying the x and y coordinates as properties.

Description Method; converts the `point` object from the movie clip's (local) coordinates, to Stage (global) coordinates.

Player Flash 5 or later.

Example The following example converts x and y coordinates of the `point` object, from the movie clip's coordinates (local) to the Stage coordinates (global). The local x and y coordinates are specified using `xmouse` and `ymouse` to retrieve the x and y coordinates of the mouse position.

```
onClipEvent(mouseMove) { point = new object(); point.x = _xmouse;
point.y = _ymouse; _root.out3 = point.x + " === " + point.y;
_root.out = _root._xmouse + " === " + _root._ymouse;
localToGlobal(point); _root.out2 = point.x + " === " + point.y;
updateAfterEvent(); }
```

See also

[MovieClip.globalToLocal](#)

CONTENTS 

Math.log

Syntax

```
Math.log(x) ;
```

Arguments x A number or expression with a value greater than 0.

Description Method; returns the natural logarithm of the argument x.

Player Flash 5 or later. In the Flash 4 Player, the methods and properties of the Math object are emulated using approximations and may not be as accurate as the non-emulated math functions supported by the Flash 5 Player.

[CONTENTS](#) 

Math.LOG2E

Syntax

Math.LOG2E

Arguments None.

Description Constant; a mathematical constant for the base-2 logarithm of the constant e (Math.E), expressed as loge2, with an approximate value of 1.442695040888963387.

Player Flash 5 or later. In the Flash 4 Player, the methods and properties of the Math object are emulated using approximations and may not be as accurate as the non-emulated math functions supported by the Flash 5 Player.

[CONTENTS](#) 

Math.LOG10E

Syntax

Math.LOG10E

Arguments None.

Description Constant; a mathematical constant for the base-10 logarithm of the constant e (Math.E), expressed as $\log_{10}e$, with an approximate value of 0.43429448190325181667.

Player Flash 5 or later. In the Flash 4 Player, the methods and properties of the Math object are emulated using approximations and may not be as accurate as the non-emulated math functions supported by the Flash 5 Player.

[CONTENTS](#) 

Math (object)

The Math object is a top-level object that you can access without using a constructor.

Use the methods and properties of this object to access and manipulate mathematical constants and functions. All of the properties and methods of the Math object are static, and must be called using the syntax

`Math.method(argument)` or `Math.constant`. In ActionScript, constants are defined with the maximum precision of double-precision IEEE-754 floating-point numbers.

The Math object is fully supported in the Flash 5 Player. In the Flash 4 Player, methods of the Math object work, but they are emulated using approximations and may not be as accurate as the non-emulated math functions supported by the Flash 5 Player.

Several of the Math object methods take the radian of an angle as an argument. You can use the equation below to calculate radian values, or simply pass the equation (entering a value for degrees) for the radian argument.

To calculate a radian value, use this formula:

$$\text{radian} = \text{Math.PI} / 180 * \text{degree}$$

The following is an example of passing the equation as an argument to calculate the sine of a 45-degree angle:

`Math.SIN(Math.PI/180 * 45)` is the same as `Math.SIN(.7854)`

Method summary for the Math object

<code>abs</code>	Computes an absolute value.
<code>acos</code>	Computes an arc cosine.
<code>asin</code>	Computes an arc sine.
<code>atan</code>	Computes an arc tangent.
<code>atan2</code>	Computes an angle from the x-axis to the point.
<code>ceil</code>	Rounds a number up to the nearest integer.
<code>cos</code>	Computes a cosine.
<code>exp</code>	Computes an exponential value.
<code>floor</code>	Rounds a number down to the nearest integer.
<code>log</code>	Computes a natural logarithm.
<code>max</code>	Returns the larger of the two integers.
<code>min</code>	Returns the smaller of the two integers.
<code>pow</code>	Computes x raised to the power of the y .
<code>random</code>	Returns a pseudo-random number between 0.0 and 1.0.
<code>round</code>	Rounds to the nearest integer.
<code>sin</code>	Computes a sine.
<code>sqrt</code>	Computes a square root.

tan	Computes a tangent.
-----	---------------------

Property summary for the Math object

All of the properties for the Math object are constants.

E	Euler's constant and the base of natural logarithms (approximately 2.718).
LN2	The natural logarithm of 2 (approximately 0.693).
LOG2E	The base 2 logarithm of e (approximately 1.442).
LN10	The natural logarithm of 10 (approximately 2.302).
LOG10E	The base 10 logarithm of e (approximately 0.434).
PI	The ratio of the circumference of a circle to its diameter (approximately 3.14159).
SQRT1_2	The reciprocal of the square root of 1/2 (approximately 0.707).
SQRT2	The square root of 2 (approximately 1.414).

Math.max

Syntax

```
Math.max(x , y);
```

Arguments x A number or expression.

y A number or expression.

Description Method; evaluates x and y and returns the larger value.

Player Flash 5 or later. In the Flash 4 Player, the methods and properties of the Math object are emulated using approximations and may not be as accurate as the non-emulated math functions supported by the Flash 5 Player.

[CONTENTS](#) 

Number.MAX_VALUE

Syntax

Number.MAX_VALUE

Arguments None.

Description Property; the largest representable number (double-precision IEEE-754). This number is approximately 1.79E+308.

Player Flash 5 or later.

[CONTENTS](#) 

Math.min

Syntax

```
Math.min(x , y);
```

Arguments x A number or expression.

y A number or expression.

Description Method; evaluates x and y and returns the smaller value.

Player Flash 5 or later. In the Flash 4 Player, the methods and properties of the Math object are emulated using approximations and may not be as accurate as the non-emulated math functions supported by the Flash 5 Player.

[CONTENTS](#) 

Number.MIN_VALUE

Syntax

Number.MIN_VALUE

Arguments None.

Description Property; the smallest representable number (double-precision IEEE-754). This number is approximately 5e-324.

Player Flash 5 or later.

[CONTENTS](#) 

Mouse (object)

Use the methods of the Mouse object to hide and show the cursor in the movie. The mouse pointer is visible by default, but you can hide it and implement a custom cursor that you create using a movie clip.

Mouse method summary

hide	Hides the cursor in the movie.
show	Displays the cursor in the movie.

[CONTENTS](#) 

MovieClip (object)

The methods for the MovieClip object provide the same functionality as the standard actions that target movie clips. There are also additional methods that provide functionality that is not available using the standard actions listed in the Actions category of the Actions panel. You do not need to use a constructor method in order to call the methods of the MovieClip object; instead, you reference movie clip instances by name, using the following syntax:

```
anyMovieClip.play();  
anyMovieClip.gotoAndPlay(3);
```

Method summary for the MovieClip object

attachMovie	Attaches a movie in the library.
duplicateMovieClip	Duplicates the specified movie clip.
getBounds	Returns the minimum and maximum x and y coordinates of a movie in a specified coordinate space.
getBytesLoaded	Returns the number of bytes loaded for the specified movie clip.
getBytesTotal	Returns the size of the movie clip in bytes.
getURL	Retrieves a document from a URL.
globalToLocal	Converts the point object from Stage coordinates to the local coordinates of the specified movie clip.
gotoAndPlay	Sends the playhead to a specific frame in the movie clip and plays the movie.
gotoAndStop	Sends the playhead to a specific frame in the movie clip and stops the movie.
hitTest	Returns <code>true</code> if bounding box of the specified movie clip intersects the bounding box of the target movie clip.
loadMovie	Loads the specified movie into the movie clip.
loadVariables	Loads variables from a URL or other location into the movie clip.
localToGlobal	Converts a Point object from the local coordinates of the movie clip to the global Stage coordinates.
nextFrame	Sends the playhead to the next frame of the movie clip.
play	Plays the specified movie clip.
prevFrame	Sends the playhead to the previous frame of the movie clip.
removeMovieclip	Removes the movie clip from the Timeline if it was created with a duplicateMovieClip action or method or the attachMovie method.
startDrag	Specifies a movie clip as draggable and begins dragging the movie clip.
stop	Stops the currently playing movie.
stopDrag	Stops the dragging of any movie clip that is being dragged.

swapDepths	Swaps the depth level of specified movie with the movie at a specific depth level.
unloadMovie	Removes a movie loaded with loadMovie.

CONTENTS 

©1995-2001 Macromedia, Inc. [All rights reserved.](#)

[Privacy policy](#) | [Contact us](#) | [Feedback](#)

NaN

Syntax

NaN

Arguments None.

Description Variable; a predefined variable with the IEEE-754 value for NaN (Not a Number).

Player Flash 5 or later.

[CONTENTS](#) 

Number.NaN

Syntax

Number.NaN

Arguments None.

Description Property; the IEEE-754 value representing Not A Number (NaN).

Player Flash 5 or later.

[CONTENTS](#) 

Number.NEGATIVE_INFINITY

Syntax

Number.NEGATIVE_INFINITY

Arguments None.

Description Property; returns the IEEE-754 value representing negative infinity. This value is the same as the global variable Infinity.

Negative infinity is a special numeric value that is returned when a mathematical operation or function returns a negative value larger than can be represented.

Player Flash 5 or later.

[CONTENTS](#) 

MovieClip.nextFrame

Syntax

```
anyMovieClip.nextFrame();
```

Arguments None.

Description Method; sends the playhead to the next frame of the movie clip.

Player Flash 5 or later.

[CONTENTS](#) 

XML.nextSibling

Syntax

```
myXML.nextSibling;
```

Arguments None.

Description Property (read-only); evaluates the XML object and references the next sibling in the parent node's child list. This method returns null if the node does not have a next sibling node. This is a read-only property and cannot be used to manipulate child nodes. Use the methods `appendChild`, `insertBefore`, and `removeNode` to manipulate child nodes.

Player Flash 5 or later.

See also

[XML.appendChild](#) [XML.insertBefore](#) [XML.removeNode](#)

CONTENTS 

XML.nodeName

Syntax

```
myXML.nodeName ;
```

Arguments None.

Description Property; takes or returns the node name of the XML object. If the XML object is an XML element (nodeType == 1), nodeName is the name of the tag representing the node in the XML file. For example, TITLE is the nodeName of an HTML TITLE tag. If the XML object is a text node (nodeType == 3), the nodeName is null.

Player Flash 5 or later.

See also

[XML.nodeType](#)

CONTENTS 

XML.nodeType

Syntax

```
myXML.nodeType ;
```

Arguments None.

Description Property (read-only); takes or returns a nodeType value, where 1 is a XML element and 3 is a text node.

Player Flash 5 or later.

See also

[XML.nodeValue](#)

[CONTENTS](#) 

XML.nodeValue

Syntax

```
myXML.nodeValue;
```

Arguments None.

Description Property; returns the node value of the XML object. If the XML object is a text node, the nodeType is 3, and the nodeValue is the text of the node. If the XML object is an XML element, it has a null nodeValue and is read-only.

Player Flash 5 or later.

See also

[XML.nodeType](#)

[CONTENTS](#) 

XMLSocket.onClose

Syntax

```
myXMLSocket.onClose( ) ;
```

Arguments None.

Description Method; a callback function that is invoked only when an open connection is closed by the server. The default implementation of this method performs no actions. To override the default implementation, you must assign a function containing your own actions.

Player Flash 5 or later.

See also

[function](#)

[XMLSocket.onConnect](#)

[CONTENTS](#) 

XMLSocket.onConnect

Syntax `myXMLSocket.onConnect (success) ;`

Arguments `success` A Boolean value indicating whether a socket connection was successfully established (true or false).

Description Method; a callback function invoked by the Flash Player when a connection request initiated through the `XMLSocket.connect` method has succeeded or failed. If the connection succeeded, the `success` argument is true; otherwise the `success` argument is false.

The default implementation of this method performs no actions. To override the default implementation, you must assign a function containing your own actions.

Player Flash 5 or later.

Example The following example illustrates the process of specifying a replacement function for the `onConnect` method in a simple chat application.

The function controls which screen the users are taken to, depending on whether a connection is successfully established. If the connection is successfully established, users are taken to the main chat screen on the frame labeled `startChat`. If the connection is not successful, users go to a screen with troubleshooting information on the frame labeled `connectionFailed`.

```
function myOnConnect(success) {  
    if (success) {  
        gotoAndPlay("startChat")  
    } else {  
        gotoAndStop("connectionFailed")  
    }  
}
```

After creating the `XMLSocket` object using the constructor method, the script installs the `onConnect` method using the assignment operator:

```
socket = new XMLSocket() socket.onConnect = myOnConnect
```

Finally, the connection is initiated. If `connect` returns false, the movie is sent directly to the frame labeled `connectionFailed`, and `onConnect` is never invoked. If `connect` returns true, the movie jumps to a frame labeled `waitForConnection`, which is the "Please wait" screen. The movie remains on the `waitForConnection` frame until the `onConnect` handler is invoked, which happens at some point in the future depending on network latency.

```
if (!socket.connect(null, 2000)) {  
    gotoAndStop("connectionFailed")  
} else {  
    gotoAndStop("waitForConnection")  
}
```

See also

[XMLSocket.connect function](#)

CONTENTS 

XML.onLoad

Syntax

```
myXML.onLoad(success);
```

Arguments `success` A boolean value indicating whether the XML object was successfully loaded with a `XML.load` or `XML.sendAndLoad` operation.

Description Method; invoked by the Flash Player when an XML document is received from the server. If the XML document is received successfully, the `success` argument is `true`. If the document was not received, or if an error occurred in receiving the response from the server, the `success` argument is `false`. The default implementation of this method is not active. To override the default implementation, you must assign a function containing your own actions.

Player Flash 5 or later.

Example The following example creates a simple Flash movie for a simple e-commerce storefront application. We use the `sendAndLoad` method to transmit an XML element containing the user's name and password, and install an `onLoad` handler to handle the reply from the server.

```
var myLoginReply = new XML();
myLoginReply.onLoad = myOnLoad;
myXML.sendAndLoad("http://www.samplestore.com/login.cgi",
                  myLoginReply);

function myOnLoad(success) {
    if (success) {
        if (e.firstChild.nodeName == "LOGINREPLY" &&
            e.firstChild.attributes.status == "OK") {
            gotoAndPlay("loggedIn")
        } else {
            gotoAndStop("loginFailed")
        }
    } else {
        gotoAndStop("connectionFailed")
    }
}
```

See also

[function](#)

[XML.load](#)

[XML.sendAndLoad](#)

[CONTENTS](#) 

XMLSocket.onXML

Syntax

```
myXMLSocket.onXML(object);
```

Argument `object` An instance of the XML object containing a parsed XML document received from a server.

Description Method; a callback function invoked by the Flash Player when the specified XML object containing an XML document arrives over an open XMLSocket connection. An XMLSocket connection may be used to transfer an unlimited number of XML documents between the client and the server. Each document is terminated with a zero byte. When the Flash Player receives the zero byte, it parses all of the XML received since the previous zero byte, or since the connection was established if this is the first message received. Each batch of parsed XML is treated as a single XML document and passed to the onXML method.

The default implementation of this method performs no actions. To override the default implementation, you must assign a function containing actions that you define.

Player Flash 5 or later.

Example The following function overrides the default implementation of the onXML method in a simple chat application. The function myOnXML instructs the chat application to recognize a single XML element, MESSAGE, in the following format:

```
<MESSAGE USER="John" TEXT="Hello, my name is John!" />.
```

The onXML handler must first be installed in the XMLSocket object as follows:

```
socket.onXML = myOnXML;
```

The function displayMessage is assumed to be a user-defined function that displays the message received to the user.

```
function myOnXML(doc) {  
    var e = doc.firstChild;  
    if (e != null && e.nodeName == "MESSAGE") {  
        displayMessage(e.attributes.user, e.attributes.text);  
    }  
}
```

See also

[function](#)

[CONTENTS](#) 

XML.parentNode

Syntax

```
myXML.parentNode;
```

Arguments None.

Description Property (read-only); references the parent node of the specified XML object, or returns null if the node has no parent. This is a read-only property and cannot be used to manipulate child nodes; use the methods `appendChild`, `insertBefore`, and `removeNode` to manipulate children.

Player Flash 5 or later.

[CONTENTS](#) 

XML.parseXML

Syntax

```
myXML.parseXML( source );
```

Arguments `source` The XML text to be parsed and passed to the specified XML object.

Description Method; parses the XML text specified in the `source` argument, and populates the specified XML object with the resulting XML tree. Any existing trees in the XML object are discarded.

Player Flash 5 or later.

[CONTENTS](#) 

Key.PGDN

Syntax

Key . PGDN

Arguments None.

Description Property; constant associated with the key code value for the Page Down key (34).

Player Flash 5 or later.

[CONTENTS](#) 

Key.PGUP

Syntax

Key . PGUP

Arguments None.

Description Property; constant associated with the key code value for the Page Up key (33).

Player Flash 5 or later.

[CONTENTS](#) 

Math.PI

Syntax

Math.PI

Arguments None.

Description Constant; a mathematical constant for the ratio of the circumference of a circle to its diameter, expressed as pi, with a value of 3.14159265358979.

Player Flash 5 or later. In the Flash 4 Player, the methods and properties of the Math object are emulated using approximations and may not be as accurate as the non-emulated math functions supported by the Flash 5 Player.

[CONTENTS](#) 

MovieClip.play

Syntax

```
anyMovieClip.play();
```

Arguments None.

Description Method; plays the movie clip.

Player Flash 5 or later.

Array.pop

x208A2 | IDS_ACTIONHELP_ARRAY_POP, Array.pop

Syntax

```
myArray.pop();
```

Arguments None.

Description Method; removes the last element from an array and returns the value of that element.

Player Flash 5 or later.

Example The following code creates the myPets array containing four elements, then removes its last element:

```
myPets = [ "cat", "dog", "bird", "fish" ]; popped = myPets.pop( );
```

[CONTENTS](#) 

Number.POSITIVE_INFINITY

Syntax

Number.POSITIVE_INFINITY

Arguments None.

Description Property; returns the IEEE-754 value representing positive infinity. This value is the same as the global variable Infinity.

Positive infinity is a special numeric value that is returned when a mathematical operation or function returns a value larger than can be represented.

Player Flash 5 or later.

[CONTENTS](#) 

Math.pow

Syntax

```
Math.pow(x , y) ;
```

Arguments x A number to be raised to a power.

y A number specifying a power the argument x is raised to.

Description Method; computes and returns x to the power of y, xy.

Player Flash 5 or later. In the Flash 4 Player, the methods and properties of the Math object are emulated using approximations and may not be as accurate as the non-emulated math functions supported by the Flash 5 Player.

[CONTENTS](#) 

MovieClip.prevFrame

Syntax

```
anyMovieClip.prevFrame();
```

Arguments None.

Description Method; sends the playhead to the previous frame and stops it.

Player Flash 5 or later.

[CONTENTS](#) 

XML.previousSibling

Syntax

```
myXML.previousSibling;
```

Description Property (read-only); evaluates the XML object and references the previous sibling in the parent node's child list. Returns null if the node does not have a previous sibling node. This is a read-only property and cannot be used to manipulate child nodes; use the methods `appendChild`, `insertBefore`, and `removeNode` to manipulate child nodes.

Player Flash 5 or later.

[CONTENTS](#) 

Array.push

x208A3 | IDS_ACTIONHELP_ARRAY_PUSH, Array.push

Syntax

```
myArray.push(value,...);
```

Arguments `value` One or more values to append to the array.

Description Method; adds one or more elements to the end of an array and returns the array's new length.

Player Flash 5 or later.

Example The following code creates the myPets array containing two elements, then adds two elements to it. After the code executes, pushed contains 4.

```
myPets = ["cat", "dog"]; pushed = myPets.push("bird", "fish");
```

[CONTENTS](#) 

Math.random

Syntax

```
Math.random( ) ;
```

Arguments None.

Description Method; returns a pseudo-random number between 0.0 and 1.0.

Player Flash 5 or later. In the Flash 4 Player, the methods and properties of the Math object are emulated using approximations and may not be as accurate as the non-emulated math functions supported by the Flash 5 Player.

See also

[random](#)

CONTENTS 

removeMovieClip

Syntax

```
removeMovieClip(target);
```

Arguments target The target path of a movie clip instance created with `duplicateMovieClip`, or the instance name of a movie clip created with the `attachMovie` or `duplicateMovie` methods of the `MovieClip` object.

Description Action; deletes a movie clip instance that was created with the `attachMovie` or `duplicateMovieClip` methods of the `MovieClip` object, or with the `duplicateMovieClip` action.

Player Flash 4 or later.

See also

[duplicateMovieClip](#) [MovieClip.duplicateMovieClip](#)
[MovieClip.attachMovie](#) [MovieClip.removeMovieClip](#)

CONTENTS 

MovieClip.removeMovieClip

Syntax

```
anyMovieClip.removeMovieClip();
```

Arguments None.

Description Method; removes a movie clip instance created with the duplicateMovieclip action, or the duplicateMovieClip or attachMovie methods of the MovieClip object.

Player Flash 5 or later.

See also

[MovieClip.loadMovie](#) [MovieClip.attachMovie](#)

[CONTENTS](#) 

XML.removeNode

Syntax

```
myXML.removeNode ( ) ;
```

Arguments None.

Description Method; removes the specified XML object from its parent.

Player Flash 5 or later.

[CONTENTS](#) 

Array.reverse

Syntax

```
myArray.reverse( );
```

Arguments None.

Description Method; reverses the array in place.

Player Flash 5 or later.

Example The following is an example of using the Array.reverse method:

```
var numbers = [1, 2, 3, 4, 5, 6]; trace(numbers.join());  
numbers.reverse(); trace(numbers.join());
```

Output:

```
1,2,3,4,5,6 6,5,4,3,2,1
```

[CONTENTS](#) 

Key.RIGHT

Syntax

Key.RIGHT

Arguments None.

Description Property; constant associated with the key code value for the Right Arrow key (39).

Player Flash 5 or later.

[CONTENTS](#) 

Math.round

Syntax

`Math.round(x) ;`

Arguments x Any number.

Description Method; rounds the value of the argument x up or down to the nearest integer and returns the value.

Player Flash 5 or later. In the Flash 4 Player, the methods and properties of the Math object are emulated using approximations and may not be as accurate as the non-emulated math functions supported by the Flash 5 Player.

[CONTENTS](#) 

Selection (object)

The Selection object allows you to set and control the currently focused editable text field. The currently focused editable text field is the field where the user's mouse pointer is currently placed. Selection-span indexes are zero-based (where the first position is 0, the second position is 1, and so on).

There is no constructor method for the Selection object, as there can only be one currently focused field at a time.

Method summary for the Selection object

<code>getBeginIndex</code>	Returns the index at the beginning of selection span. Returns -1 if there is no index or currently selected field.
<code>getCaretIndex</code>	Returns the current caret position in the currently focused selection span. Returns -1 if there is no caret position or currently focused selection span.
<code>getEndIndex</code>	Returns the index at the end of the selection span. Returns -1 if there is no index or currently selected field.
<code>getFocus</code>	Returns the name of the variable for currently focused editable text field. Returns null if there is no currently focused editable text field.
<code>setFocus</code>	Focuses the editable text field associated with the variable specified in the argument.
<code>setSelection</code>	Sets the beginning and ending indexes of the selection span.

XML.send

Syntax

```
myXML.send(url);  
myXML.send(url, window);
```

Arguments `url` The destination URL for the specified XML object.

`window` The browser window to display data returned by the server: `_self` specifies the current frame in the current window, `_blank` specifies a new window, `_parent` specifies the parent of the current frame, and `_top` specifies the top-level frame in the current window.

Description Method; encodes the specified XML object into a XML document and sends it to the specified URL using the POST method.

Player Flash 5 or later.

[CONTENTS](#) 

XMLSocket.send

Syntax

```
myXMLSocket.send(object);
```

Arguments `object` An XML object or other data to transmit to the server.

Description Method; converts the XML object or data specified in the `object` argument to a string and transmits it to the server, followed by a zero byte. If `object` is an XML object, the string is the XML textual representation of the XML object. The send operation is asynchronous; it returns immediately, but the data may be transmitted at a later time. The XMLSocket.send method does not return a value indicating whether the data was successfully transmitted.

If the `myXMLSocket` object is not connected to the server (using `XMLSocket.connect`), the XMLSocket.send operation will fail.

Player Flash 5 or later.

Example The following example illustrates how you could specify a user name and password to send the XML object `myXML` to the server:

```
var myXML = new XML(); var myLogin = myXML.createElement("login");
myLogin.attributes.username = usernameTextField;
myLogin.attributes.password = passwordTextField;
myXML.appendChild(myLogin); myXMLSocket.send(myXML);
```

See also

[XMLSocket.connect](#)

CONTENTS 

XML.sendAndLoad

Syntax

```
myXML.sendAndLoad(url,targetXMLObject);
```

Arguments `url` The destination URL for the specified XML object. The URL must be in the same subdomain as the URL where the movie was downloaded from.

`targetXMLObject` An XML object created with the XML constructor method that will receive the return information from the server.

Description Method; encodes the specified XML object into a XML document, sends it to the specified URL using the POST method, downloads the server's response and then loads it into the `targetXMLObject` specified in the arguments. The server response is loaded in the same manner used by the load method.

Player Flash 5 or later.

See also

[XML.load](#)

CONTENTS 

Date.setDate

Syntax

```
myDate.setDate(date);
```

Arguments date A integer from 1 to 31.

Description Method; sets the day of the month for the specified Date object, according to local time. Local time is determined by the operating system on which the Flash Player is running.

Player Flash 5 or later.

[CONTENTS](#) 

Selection.setFocus

Syntax

```
Selection.setFocus(variable);
```

Arguments variable A string specifying the name of a variable associated with a text field using dot or slash notation.

Description Method; focuses the editable text field associated with the specified variable.

Player Flash 5 or later.

[CONTENTS](#) 

Date.setFullYear

Syntax

```
myDate.setFullYear(year [, month [, date]] );
```

Arguments **year** A four-digit number specifying a year. Two-digit numbers do not represent years; for example, 99 is not the year 1999, but the year 99.

month An integer from 0 (January) to 11 (December). This argument is optional.

date A number from 1 to 31. This argument is optional.

Description Method; sets the year of the specified Date object, according to local time. If the **month** and **date** arguments are specified, they are also set to local time. Local time is determined by the operating system on which the Flash Player is running.

The results of `getUTCDay` and `getDay` may change as a result of calling this method.

Player Flash 5 or later.

[CONTENTS](#) 

Date.setHours

Syntax

```
myDate.setHours(hour);
```

Arguments hour An integer from 0 (midnight) to 23 (11 p.m.).

Description Method; sets the hours for the specified Date object according to local time. Local time is determined by the operating system on which the Flash Player is running.

Player Flash 5 or later.

[CONTENTS](#) 

Date.setMilliseconds

Syntax

```
myDate.setMilliseconds(milliseconds);
```

Arguments milliseconds An integer from 0 to 999.

Description Method; sets the milliseconds for the specified Date object according to local time. Local time is determined by the operating system on which the Flash Player is running.

Player Flash 5 or later.

[CONTENTS](#) 

Date.setMinutes

Syntax

```
myDate.setMinutes(minute);
```

Arguments minute An integer from 0 to 59.

Description Method; sets the minutes for a specified Date object according to local time. Local time is determined by the operating system on which the Flash Player is running.

Player Flash 5 or later.

[CONTENTS](#) 

Date.setMonth

Syntax

```
myDate.setMonth(month [ , date ] );
```

Arguments month An integer from 0 (January) to 11 (December).

date An integer from 1 to 31. This argument is optional.

Description Method; sets the month for the specified Date object in local time. Local time is determined by the operating system on which the Flash Player is running.

Player Flash 5 or later.

[CONTENTS](#) 

Sound.setPan

Syntax

```
mySound.setPan(pan);
```

Arguments `pan` An integer specifying the left-right balance for a sound. The range of valid values is -100 to 100, where -100 uses only the left channel, 100 uses only the right channel, and 0 balances the sound evenly between the two channels.

Description Method; determines how the sound is played in the left and right channels (speakers). For mono sounds, `pan` affects which speaker (left or right) the sound plays through.

This method is cumulative with the `setVolume` and `setTransform` methods, and calling this method deletes and updates previous `setPan` and `setTransform` settings.

Player Flash 5 or later.

Example The following example uses `setVolume` and `setPan` to control a sound object with the specified target "u2":

```
onClipEvent(mouseDown) { // create a sound object and s = new
Sound(this); // attach a sound in the library s.attachSound("u2");
//set volume at 50% s.setVolume(50); //turn off the sound in the
right channel s.setPan(-100); //start 30 seconds into the sound and
play it 5 times s.start(30, 5);
```

See also

[Sound.setTransform](#) [Sound.setVolume](#)

CONTENTS 

Date.setSeconds

Syntax

```
myDate.setSeconds ( second ) ;
```

Arguments `second` An integer from 0 to 59.

Description Method; sets the seconds for the specified Date object in local time. Local time is determined by the operating system on which the Flash Player is running.

Player Flash 5 or later.

[CONTENTS](#) 

Selection.setSelection

Syntax

```
Selection.setSelection(start, end);
```

Arguments `start` The beginning index of the selection span.

`end` The ending index of the selection span.

Description Method; sets the selection span of the currently focused text field. The new selection span will begin at the index specified in the `start` argument, and end at the index specified in the `end` argument. Selection span indexes are zero-based (where the first position is 0, the second position is 1, and so on). This method has no effect if there is no currently focused text field.

Player Flash 5 or later.

[CONTENTS](#) 

Date.setTime

Syntax

```
myDate.setTime(milliseconds) ;
```

Arguments milliseconds An integer from 0 to 999.

Description Method; sets the Date for the specified Date object in milliseconds.

Player Flash 5 or later.

[CONTENTS](#) 

Color.setTransform

Syntax

```
myColor.setTransform(colorTransformObject);
```

Arguments `colorTransformObject` An object created using the constructor of the generic `Object` object, specifying color transform values for parameters. The color transform object must have the parameters `ra`, `rb`, `ga`, `gb`, `ba`, `bb`, `aa`, `ab`, which are explained below.

Description Method; sets color transform information for a `Color` object. The `colorTransformObject` argument is an object that you create using the generic `Object` object with parameters specifying the percentage and offset values for the red, green, blue, and alpha (transparency) components of a color, entered in a `0xRRGGBBAA` format.

The parameters for a color transform object are defined as follows:

- ▶ `ra` is the percentage for the red component (-100 to 100).
- ▶ `rb` is the offset for the red component (-255 to 255).
- ▶ `ga` is the percentage for the green component (-100 to 100).
- ▶ `gb` is the offset for the green component (-255 to 255).
- ▶ `ba` is the percentage for the blue component (-100 to 100).
- ▶ `bb` is the offset for the blue component (-255 to 255).
- ▶ `aa` is the percentage for alpha (-100 to 100).
- ▶ `ab` is the offset for alpha (-255 to 255).

You create a color transform object as follows:

```
myColorTransform = new Object(); myColorTransform.ra = 50;  
myColorTransform.rb = 244; myColorTransform.ga = 40;  
myColorTransform.gb = 112; myColorTransform.ba = 12;  
myColorTransform.bb = 90; myColorTransform.aa = 40;  
myColorTransform.ab = 70;
```

You could also use the following syntax:

```
myColorTransform = { ra: `50`, rb: `244`, ga: `40`, gb: `112`, ba:  
`12`, bb: `90`, aa: `40`, ab: `70` }
```

Player Flash 5 or later.

Example The following example shows the process of creating a new `Color` object for a target movie, creating a color transform object with the parameters defined above using the `Object` constructor, and passing the color transform object to a `Color` object using the `setTransform` method.

```
//Create a color object called myColor for the target myMovie  
myColor = new Color(myMovie); //Create a color transform object  
called myColorTransform using //the generic Object object  
myColorTransform = new Object; // Set the values for  
myColorTransform myColorTransform = { ra: '50', rb: '244', ga:  
'40', gb: '112', ba: '12', bb: '90', aa: '40', ab: '70'}  
//Associate the color transform object with the Color object  
created for myMovie myColor.setTransform(myColorTransform);
```

©1995-2001 Macromedia, Inc. [All rights reserved.](#)

[Privacy policy](#) | [Contact us](#) | [Feedback](#)

Sound.setTransform

Syntax

```
mySound.setTransform( soundTransformObject );
```

Arguments `soundTransformObject` An object created with the constructor for the generic `Object` object.

Description Method; sets the sound transform information for a `Sound` object. This method is cumulative with the `setVolume` and `setPan` methods, and calling this method deletes and updates any previous `setPan` or `setVolume` settings. This call is for expert users who want to add interesting effects to sounds.

Sounds use a considerable amount of disk space and memory. Because stereo sounds use twice as much data as mono sounds, it's generally best to use 22-Khz 6-bit mono sounds. You can use the `setTransform` method to play mono sounds as stereo, play stereo sounds as mono, and to add interesting effects to sounds.

The `soundTransformObject` argument is an object that you create using the constructor method of the generic `Object` object with parameters specifying how the sound is distributed to the left and right channels (speakers).

The parameters for the `soundTransformObject` are as follows:

`l1` A percentage value specifying how much of the left input to play in the left speaker (-100 to 100).

`lr` A percentage value specifying how much of the right input to play in the left speaker (-100 to 100).

`rr` A percentage value specifying how much of the right input to play in the right speaker (-100 to 100).

`rl` A percentage value specifying how much of the left input to play in the right speaker (-100 to 100).

The net result of the parameters is represented by the following formula:

$$\text{leftOutput} = \text{left input} * l1 + \text{right input} * lr$$
$$\text{rightOutput} = \text{right input} * rr + \text{left input} * rl$$

The values for left input or right input are determined by the type (stereo or mono) of sound in your movie.

Stereo sounds divide the sound input evenly between the left and right speakers and have the following transform settings by default:

$$l1 = 100 \quad lr = 0 \quad rr = 100 \quad rl = 0$$

Mono sounds play all sound input in the left speaker and have the following transform settings by default:

$$l1 = 100 \quad lr = 100 \quad rr = 0 \quad rl = 0$$

Player Flash 5 or later.

Example The following example creates a sound transform object that plays both the left and right channels in the left channel:

```
mySoundTransformObject = new Object
mySoundTransformObject.l1 = 100
mySoundTransformObject.lr = 100
mySoundTransformObject.rr = 0
mySoundTransformObject.rl = 0
```

In order to apply the sound transform object to a `Sound` object, you need to pass the object to the `Sound` object using `setTransform` as follows:

```
mySound.setTransform(mySoundTransformObject);
```

The following are examples of settings that can be set using `setTransform`, but cannot be set using `setVolume` or `setPan`, even if combined.

This code plays both the left and right channels through the left channel:

```
mySound.setTransform( soundTransformObjectLeft );
```

In the above code, the `soundTransformObjectLeft` has the following parameters:

```
l1 = 100 lr = 100 rr = 0 rl = 0
```

This code plays a stereo sound as mono:

```
setTransform( soundTransformObjectMono );
```

In the above code, the `soundTransformObjectMono` has the following parameters:

```
l1 = 50 lr = 50 rr = 50 rl = 50
```

This code plays the left channel at half capacity and adds the rest of the left channel to the right channel:

```
setTransform( soundTransformObjectHalf );
```

In the above code, the `soundTransformObjectHalf` has the following parameters:

```
l1 = 50 lr = 0 rr = 100 rl = 50
```

See also [Constructor for the Object object](#)

[CONTENTS](#) 

Date.setUTCDate

Syntax

```
myDate.setUTCDate(date);
```

Arguments `date` An integer from 1 to 31.

Description Method; sets the date for the specified Date object in universal time. Calling this method does not modify the other fields of the specified Date, but the `getUTCDay` and `getDay` methods may report a new value if the day of the week changes as a result of calling this method.

Player Flash 5 or later.

[CONTENTS](#) 

Date.setUTCFullYear

Syntax

```
myDate.setUTCFullYear(year [, month [, date]]);
```

Arguments year The year specified as a full four-digit year, for example, 2000.

month An integer from 0 (January) to 11 (December). This argument is optional.

date An integer from 1 to 31. This argument is optional.

Description Method; sets the year or the specified Date object (*mydate*) in universal time.

Optionally, this method can also set the month and date represented by the specified Date object. No other fields of the Date object are modified. Calling `setUTCFullYear` may cause `getUTCDay` and `getDay` to report a new value if the day of the week changes as a result of this operation.

Player Flash 5 or later.

[CONTENTS](#) 

Date.setUTCHours

Syntax

```
myDate.setUTCHours(hour [ , minute [ , second [ , millisecond]]]);
```

Arguments hour An integer from 0 (midnight) to 23 (11p.m.).

minute An integer from 0 to 59. This argument is optional.

second An integer from 0 to 59. This argument is optional.

millisecond An integer from 0 to 999. This argument is optional.

Description Method; sets the hour for the specified Date object in universal time.

Player Flash 5 or later.

[CONTENTS](#) 

Date.setUTCMilliseconds

Syntax

```
myDate.setUTCMilliseconds(milliseconds);
```

Arguments milliseconds An integer from 0 to 999.

Description Method; sets the milliseconds for the specified Date object in universal time.

Player Flash 5 or later.

[CONTENTS](#) 

Date.setUTCMinutes

Syntax

```
myDate.setUTCMinutes(minute [ , second [ , millisecond]]);
```

Arguments minute An integer from 0 to 59.

second An integer from 0 to 59. This argument is optional.

millisecond An integer from 0 to 999. This argument is optional.

Description Method; sets the minute for the specified Date object in universal time.

Player Flash 5 or later.

[CONTENTS](#) 

Date.setUTCMonth

Syntax

```
myDate.setUTCMonth(month [, date]);
```

Arguments month An integer from 0 (January) to 11 (December).

date An integer from 1 to 31. This argument is optional.

Description Method; sets the month, and optionally the day (date), for the specified Date object in universal time. Calling this method does not modify the other fields of the specified Date object, but the `getUTCDay` and `getDay` methods may report a new value if the day of the week changes as a result of specifying the `date` argument when calling `setUTCMonth`.

Player Flash 5 or later.

[CONTENTS](#) 

Date.setUTCSeconds

Syntax

```
myDate.setUTCSeconds(second [ , millisecond ] );
```

Arguments `second` An integer from 0 to 59.

`millisecond` An integer from 0 to 999. This argument is optional.

Description Method; sets the seconds for the specified Date object in universal time.

Player Flash 5 or later.

[CONTENTS](#) 

Sound.setVolume

Syntax

```
mySound.setVolume(volume);
```

Arguments `volume` A number from 0 to 100 representing a volume level. 100 is full volume and 0 is no volume. The default setting is 100.

Description Method; sets the volume for the Sound object.

This method is cumulative with the `setPan` and `setTransform` methods.

Player Flash 5 or later.

Example The following example sets volume to 50% and transfers the sound over time from the left speaker to the right speaker:

```
onClipEvent (load) {  
    i = -100;  
    s = new sound();  
    s.setVolume(50);  
}  
onClipEvent (enterFrame) {  
    S.setPan(i++);  
}
```

See also

[Sound.setPan](#) [Sound.setTransform](#)

CONTENTS 

Date.setYear

Syntax

```
myDate.setYear( year );
```

Arguments year A four-digit number, for example, 2000.

Description Method; sets the year for the specified date object in local time. Local time is determined by the operating system on which the Flash Player is running.

Player Flash 5 or later.

[CONTENTS](#) 

Array.shift

Syntax

```
myArray.shift();
```

Arguments None.

Description Method; removes the first element from an array and returns that element.

Player Flash 5 or later.

Example The following code creates the array myPets and then removes the first element from the array:

```
myPets = ["cat", "dog", "bird", "fish"]; shifted = myPets.shift();
```

The return value is cat.

See also

[Array.pop](#) [Array.unshift](#)

[CONTENTS](#) 

Key.SHIFT

Syntax

Key.SHIFT

Arguments None.

Description Property; constant associated with the key code value for the Shift key (16).

Player Flash 5 or later.

[CONTENTS](#) 

Mouse.show

Syntax

```
Mouse.show( ) ;
```

Arguments None.

Description Method; makes the cursor visible in a movie. The cursor is visible by default.

Player Flash 5 or later.

See also

[_xmouse](#)

[_ymouse](#)

[Mouse.show](#)

[CONTENTS](#) 

Math.sin

Syntax

```
Math.sin(x);
```

Arguments x An angle measured in radians.

Description Method; computes and returns the sine of the specified angle in radians. Use the information outlined in the introduction to the Math object to calculate a radian.

Player Flash 5 or later. In the Flash 4 Player, the methods and properties of the Math object are emulated using approximations and may not be as accurate as the non-emulated math functions supported by the Flash 5 Player.

See also

[Math \(object\)](#)

[CONTENTS](#) 

Array.slice

Syntax

```
myArray.slice(start, end);
```

Arguments **start** A number specifying the index of the starting point for the slice. If **start** is a negative number, the starting point begins at the end of the array, where -1 is the last element.

end A number specifying the index of the ending point for the slice. If you omit this argument, the slice includes all elements from the start to the end of the array. If **end** is a negative number, the ending point is specified from the end of the array, where -1 is the last element.

Description Method; extracts a slice or a substring of the array and returns it as a new array without modifying the original array. The returned array includes the **start** element and all elements up to, but not including, the **end** element.

Player Flash 5 or later.

[CONTENTS](#) 

String.slice

Syntax

```
myString.slice(start,end);
```

Arguments **start** A number specifying the index of the starting point for the slice. If **start** is a negative number, the starting point is determined from the end of the string, where -1 is the last character.

end A number specifying the index of the ending point for the slice. If **end** is not specified, the slice includes all characters from the start to the end of the string. If **end** is a negative number, the ending point is determined from the end of the string, where -1 is the last character.

Description Method; extracts a slice, or substring, of the specified String object; then returns it as a new string without modifying the original String object. The returned string includes the **start** character and all characters up to (but not including) the **end** character.

Player Flash 5 or later.

[CONTENTS](#) 

Array.sort

Syntax

```
myArray.sort();  
myArray.sort(orderfunc);
```

Arguments `orderfunc` An optional comparison function used to determine the sorting order. Given the arguments A and B, the specified ordering function should perform a sort as follows:

- ▶ -1 if A appears before B in the sorted sequence
- ▶ 0 if A = B
- ▶ 1 if A appears after B in the sorted sequence

Description Method; sorts the array in place, without making a copy. If you omit the `orderfunc` argument, Flash sorts the elements in place using the < comparison operator.

Player Flash 5 or later.

Example The following example uses `Array.sort` without specifying the `orderfunc` argument:

```
var fruits = ["oranges", "apples", "strawberries", "pineapples",  
"cherries"]; trace(fruits.join()); fruits.sort();  
trace(fruits.join());
```

Output:

```
oranges,apples,strawberries,pineapples,cherries  
apples,cherries,oranges,pineapples,strawberries
```

The following example uses `array.sort` with a specified order function:

```
var passwords = [ "gary:foo", "mike:bar", "john:snafu", "steve:yuck",  
"daniel:1234" ]; function order (a, b) { // Entries to be sorted are  
in form // name:password // Sort using only the name part of the //  
entry as a key. var name1 = a.split(':')[0]; var name2 =  
b.split(':')[0]; if (name1 < name2) { return -1; } else if (name1 >  
name2) { return 1; } else { return 0; } } for (var i=0; i<  
passwords.length; i++) { trace (passwords.join()); }  
passwords.sort(order); trace ("Sorted:"); for (var i=0; i<  
password.length; i++) { trace (passwords.join()); }
```

Output:

```
daniel:1234 gary:foo john:snafu mike:bar steve:yuck
```

[CONTENTS](#) 

Key.SPACE

Syntax

Key . SPACE

Arguments None.

Description Property; constant associated with the key code value for the Spacebar (32).

Player Flash 5 or later.

[CONTENTS](#) 

Array.splice

Syntax

```
myArray.splice(start, deleteCount, value0,value1...valueN);
```

Arguments `start` The index of the element in the array where the insertion and/or deletion begins.

`deleteCount` The number of elements to be deleted. This number includes the element specified in the `start` argument. If no value is specified for `deleteCount`, the method deletes all of the values from the `start` element to the last element in the array.

`value` Zero or more values to insert into the array at the insertion point specified in the `start` argument. This argument is optional.

Description Method; adds and/or removes elements from an array. This method modifies the array itself without making a copy.

Player Flash 5 or later.

[CONTENTS](#) 

String.split

Syntax

```
myString.split(delimiter);
```

Arguments `delimiter` The character used to delimit the string.

Description Method; splits a String object by breaking the string wherever the specified `delimiter` argument occurs, and returns the substrings in an array. If no delimiter is specified, the returned array contains only one element—the string itself. If the delimiter is an empty string, each character in the String object becomes an element in the array.

Player Flash 5 or later.

[CONTENTS](#) 

Math.sqrt

Syntax

```
Math.sqrt(x);
```

Arguments x Any number or expression greater than or equal to 0.

Description Method; computes and returns the square root of the specified number.

Player Flash 5 or later. In the Flash 4 Player, the methods and properties of the Math object are emulated using approximations and may not be as accurate as the non-emulated math functions supported by the Flash 5 Player.

[CONTENTS](#) 

Math.SQRT1_2

Syntax

Math.SQRT1_2

Arguments None.

Description Constant; a mathematical constant for the reciprocal of the square root of one half (1/2), with an approximate value of 0.707106781186.

Player Flash 5 or later. In the Flash 4 Player, the methods and properties of the Math object are emulated using approximations and may not be as accurate as the non-emulated math functions supported by the Flash 5 Player.

[CONTENTS](#) 

Math.SQRT2

Syntax

`Math.SQRT2`

Arguments None.

Description Constant; a mathematical constant for the square root of 2, with an approximate value of 1.414213562373.

Player Flash 5 or later. In the Flash 4 Player, the methods and properties of the Math object are emulated using approximations and may not be as accurate as the non-emulated math functions supported by the Flash 5 Player.

[CONTENTS](#) 

Sound.start

Syntax

```
mySound.start();  
mySound.start([secondOffset, loop]);
```

secondOffset An optional argument allowing you to start the sound playing at a specific point. For example, if you have a 30-second sound and want the sound to start playing in the middle, specify 15 for the **secondOffset** argument. The sound is not delayed 15 seconds, but rather starts playing at the 15-second mark.

loop An optional argument allowing you to specify the number of times the sound should loop.

Description Method; starts playing the last attached sound from the beginning if no argument is specified, or starting at the point in the sound specified by the **secondOffset** argument.

Player Flash 5 or later.

See also

[Sound.setPan](#)

[Sound.stop](#)

CONTENTS 

startDrag

Syntax

```
startDrag(target);  
startDrag(target,[lock]);  
startDrag(target [,lock [,left , top , right , bottom]]);
```

Arguments *target* The target path of the movie clip to drag.

lock A Boolean value specifying whether the draggable movie clip is locked to the center of the mouse position (*true*), or locked to the point where the user first clicked on the movie clip (*false*). This argument is optional.

left, top, right, bottom Values relative to the coordinates of the movie clip's parent that specify a constraint rectangle for the movie clip. These arguments are optional.

Description Action; makes the *target* movie clip draggable while the movie is playing. Only one movie clip can be dragged at a time. Once a *startDrag* operation is executed, the movie clip remains draggable until explicitly stopped by a *stopDrag* action, or until a *startDrag* action for another movie clip is called.

Example To create a movie clip that users can position in any location, attach the *startDrag* and *stopDrag* actions to a button inside the movie clip, as in the following:

```
on(press) {  
    startDrag("",true);  
}  
on(release) {  
    stopDrag();  
}
```

See also

[stopDrag](#)
[_droptarget](#)

[CONTENTS](#) 

MovieClip.startDrag

Syntax

```
anyMovieClip.startDrag([lock, left, right, top, bottom]);
```

Arguments *lock* A Boolean value specifying whether the draggable movie clip is locked to the center of the mouse position (*true*), or locked to the point where the user first clicked on the movie clip (*false*). This argument is optional.

left, top, right, bottom Values relative to the coordinates of the movie clip's parent that specify a constraint rectangle for the movie clip. These arguments are optional.

Description Method; allows the user to drag the specified movie clip. The movie remains draggable until explicitly stopped by calling the `stopDrag` method, or until another movie clip is made draggable. Only one movie clip is draggable at a time.

Player Flash 5 or later.

See also

[MovieClip.stopDrag](#)
[_droptarget](#)

CONTENTS 

XML.status

Syntax

```
myXML.status;
```

Arguments None.

Description Property; automatically sets and returns a numeric value indicating whether an XML document was successfully parsed into an XML object. The following is a list of the numeric status codes and a description of each:

- ▶ 0 No error; parse completed successfully.
- ▶ -2 A CDATA section was not properly terminated.
- ▶ -3 The XML declaration was not properly terminated.
- ▶ -4 The DOCTYPE declaration was not properly terminated.
- ▶ -5 A comment was not properly terminated.
- ▶ -6 An XML element was malformed.
- ▶ -7 Out of memory.
- ▶ -8 An attribute value was not properly terminated.
- ▶ -9 A start-tag was not matched with an end-tag.
- ▶ -10 An end-tag was encountered without a matching start-tag.

Player Flash 5 or later.

[CONTENTS](#) 

MovieClip.stop

Syntax

```
anyMovieClip.stop();
```

Arguments None.

Description Method; stops the movie clip currently playing.

Player Flash 5 or later.

[CONTENTS](#) 

stopDrag

Syntax

```
stopDrag( );
```

Arguments None.

Description Action; stops the current drag operation.

Player Flash 4 or later.

Example This statement stops the drag action on the instance mc when the user releases the mouse button:

```
on(press) {  
    startDrag("mc");  
}  
on(release) {  
    stopdrag();  
}
```

See also

[startDrag _droptarget](#)

[CONTENTS](#) 

MovieClip.stopDrag

Syntax

```
anyMovieClip.stopDrag();
```

Arguments None.

Description Method; ends a drag action implemented with the startDrag method. A movie remains draggable until a stopDrag method is added, or until another movie becomes draggable. Only one movie clip is draggable at a time.

Player Flash 5 or later.

See also

[_droptarget](#)

[MovieClip.startDrag](#)

[CONTENTS](#) 

" " (string delimiter)

Syntax

```
"text"
```

Arguments `text` Any text.

Description String delimiter; when used before and after a string, quotes indicate that the string is a literal—not a variable, numerical value, or other ActionScript element.

Player Flash 4 or later.

Example This statement uses quotes to indicate that the string "Prince Edward Island" is a literal string, and not the value of a variable:

```
province = "Prince Edward Island"
```

See also

[String \(object\)](#)

[String \(function\)](#)

[CONTENTS](#) 

String.substr

Syntax

```
myString.substr( start, length ) ;
```

Arguments **start** An integer that indicates the position of the first character in the substring being created. If **start** is a negative number, the starting position is determined from the end of the string, where the -1 is the last character.

length The number of characters in the substring being created. If **length** is not specified, the substring includes all of the characters from the start to the end of the string.

Description Method; returns the characters in a string from the index specified in the **start** argument through the number of characters specified in the **length** argument.

Player Flash 5 or later.

[CONTENTS](#) 

substring

Syntax

```
substring(string, index, count);
```

Arguments `string` The string from which to extract the new string.

`index` The number of the first character to extract.

`count` The number of characters to include in the extracted string, not including the index character.

Description String function; extracts part of a string.

Player Flash 4 or later. This function has been deprecated in Flash 5.

See also

[String.substring](#)

CONTENTS 

String.substring

Syntax

```
myString.substring(from,to);
```

Arguments **from** An integer that indicates the position of the first character in the substring being created. Valid values for **from** are 0 through `string.length - 1`.

to An integer that is 1+ the index of the last character in the substring being created. Valid values for **to** are 1 through `string.length`. If the **to** argument is not specified, the end of the substring is the end of the string. If **from** equals **to**, the method returns an empty string. If **from** is greater than **to**, the arguments are automatically swapped before the function executes.

Description Method; returns a string consisting of the characters between the points specified by the **from** and **to** arguments.

Player Flash 5 or later.

[CONTENTS](#) 

MovieClip.swapDepths

Syntax

```
anyMovieClip.swapDepths(depth);  
anyMovieClip.swapDepths(target);
```

Arguments **target** The movie clip instance whose depth that is being swapped by the instance specified in anyMovieClip. Both instances must have the same parent movie clip.

depth A number specifying the depth level where the anyMovieClip is to be placed.

Description Method; swaps the stacking, or z, order (depth level) of the specified instance with the movie specified by the **target** argument, or with the movie that currently occupies the **depth** level specified in the argument. Both movies must have the same parent movie clip. Swapping the depth level of movie clips has the effect of moving one movie in front of or behind the other. If a movie is tweening when this method is called, the tweening is stopped.

Player Flash 5 or later.

See also

[_level](#)

CONTENTS 

Key.TAB

Syntax

Key . TAB

Arguments None.

Description Property; constant associated with the key code value for the Tab key (9).

Player Flash 5 or later.

[CONTENTS](#) 

Math.tan

Syntax

```
Math.tan(x) ;
```

Arguments x An angle measured in radians.

Description Method; computes and returns the tangent of the specified angle. Use the information outlined in the introduction to the Math object to calculate a radian.

Player Flash 5 or later. In the Flash 4 Player, the methods and properties of the Math object are emulated using approximations and may not be as accurate as the non-emulated math functions supported by the Flash 5 Player.

[CONTENTS](#) 

String.toLowerCase

Syntax

```
myString.toLowerCase( ) ;
```

Arguments None.

Description Method; returns a copy of the String object, with all of the uppercase characters converted to lowercase.

Player Flash 5 or later.

[CONTENTS](#) 

Array.toString

Syntax

```
myArray.toString();
```

Arguments None.

Description Method; returns a string value representing the elements in the specified Array object. Every element in the array, starting with index 0 and ending with index `myArray.length-1`, is converted to a concatenated string separated by commas.

Player Flash 5 or later.

Example The following example creates `myArray` and converts it to a string:

```
myArray = new Array();  
myArray[0] = 1;  
myArray[1] = 2;  
myArray[2] = 3;  
myArray[3] = 4;  
myArray[4] = 5;  
trace(myArray.toString());
```

Output:

1,2,3,4,5

[CONTENTS](#) 

Boolean.toString

Syntax

```
Boolean.toString( ) ;
```

Arguments None.

Description Method; returns the string representation, true or false of the Boolean object.

Player Flash 5 or later.

[CONTENTS](#) 

Date.toString

Syntax

```
myDate.toString();
```

Arguments None.

Description Method; returns a string value for the specified date object in a readable format.

Player Flash 5 or later.

Example The following example returns the information in the dateOfBirth Date object as a string:

```
var dateOfBirth = new Date(74, 7, 7, 18, 15);  
trace (dateOfBirth.toString());
```

Output (for Pacific Standard Time):

Wed Aug 7 18:15:00 GMT-0700 1974

[CONTENTS](#) 

Number.toString

Syntax

```
myNumber.toString(radix);
```

Arguments `radix` Specifies the numeric base (from 2 to 36) to use for the number-to-string conversion. If you do not specify the `radix` argument, the default value is 10.

Description Method; returns the string representation of the specified Number object (`myNumber`).

Player Flash 5 or later.

Example The following example uses the `Number.toString` method, specifying 2 for the `radix` argument:

```
myNumber = new Number (1000); (1000).toString(2);
```

Returns a string containing the binary representation of the number 1000.

[CONTENTS](#) 

Object.toString

Syntax

```
myObject.toString( );
```

Arguments None.

Description Method; converts the specified object to a string, and returns it.

Player Flash 5 or later.

[CONTENTS](#) 

XML.toString

Syntax

```
myXML.toString();
```

Arguments None.

Description Method; evaluates the specified XML object, constructs a textual representation of the XML structure including the node, children, and attributes, and returns the result as a string.

For top-level XML objects (those created with the constructor), XML.toString outputs the document's XML declaration (stored in XML.xmlDecl), followed by the document's DOCTYPE declaration (stored in XML.docTypeDecl), followed by the text representation of all XML nodes in the object. The XML declaration is not output if XML.xmlDecl is undefined. The DOCTYPE declaration is not output if XML.docTypeDecl is undefined.

Player Flash 5 or later.

Example The following code is an example of the XML.toString method:

```
node = new XML("<h1>test</h1>"); trace(node.toString()); sends  
<H1>test</H1> to the output window
```

See also

[XML.xmlDecl](#) [XML.docTypeDecl](#)

[CONTENTS](#) 

String.toUpperCase

Syntax

```
myString.toUpperCase( ) ;
```

Arguments None.

Description Method; returns a copy of the String object, with all of the lowercase characters converted to uppercase.

Player Flash 5 or later.

[CONTENTS](#) 

unloadMovie

Syntax

```
unloadMovie(location);
```

Arguments `location` The depth level or target movie clip from which to unload the movie.

Description Action; removes a movie from the Flash Player that was previously loaded using the `loadMovie` action.

Player Flash 3 or later.

Example The following example unloads the main movie, leaving the Stage blank:

```
unloadMovie(_root);
```

The following example unloads the movie at level 15, when the user clicks the mouse:

```
on(press) { unloadMovie(_level15); }
```

See also

[loadMovie](#)

[CONTENTS](#) 

MovieClip.unloadMovie

Syntax

```
anyMovieClip.unloadMovie();
```

Arguments None.

Description Method; removes a movie clip loaded with the loadMovie or attachMovie MovieClip methods.

Player Flash 5 or later.

See also

[MovieClip.loadMovie](#) [MovieClip.attachMovie](#)

CONTENTS 

Key.UP

Syntax

Key.UP

Arguments None.

Description Property; constant associated with the key code value for the Up Arrow key (38).

Player Flash 5 or later.

[CONTENTS](#) 

Date.UTC

Syntax

```
Date.UTC(year, month[, date[, hour[, minute[, second[, millisecond ]]]]);
```

Arguments year A four-digit number, for example, 2000.

month An integer from 0 (January) to 11 (December).

date An integer from 1 to 31. This argument is optional.

hour An integer from 0 (midnight) to 23 (11 p.m.).

minute An integer from 0 to 59. This argument is optional.

second An integer from 0 to 59. This argument is optional.

millisecond An integer from 0 to 999. This argument is optional.

Description Method; returns the number of milliseconds between midnight on January 1, 1970, universal time, and the time specified in the arguments. This is a static method that is invoked through the Date object constructor, not through a specific Date object. This method allows you to create a Date object that assumes universal time, whereas the Date constructor assumes local time.

Player Flash 5 or later.

Example The following example creates a new Date object gary_birthday defined in universal time. This is the universal time variation of the example used for the constructor method new Date():

```
gary_birthday = new Date(Date.UTC(1974, 7, 8));
```

CONTENTS 

Boolean.valueOf

Syntax

```
Boolean.valueOf ( ) ;
```

Arguments None.

Description Method; returns the primitive value type of the specified Boolean object, and converts the Boolean wrapper object to this primitive value type.

Player Flash 5 or later.

[CONTENTS](#) 

Number.valueOf

Syntax

```
myNumber.valueOf ( ) ;
```

Arguments None.

Description Method; returns the primitive value type of the specified Number object, and converts the Number wrapper object to the primitive value type.

Player Flash 5 or later.

[CONTENTS](#) 

Object.valueOf

Syntax

```
myObject.valueOf( ) ;
```

Arguments None.

Description Method; returns the primitive value of the specified object. If the object does not have a primitive value, the object itself is returned.

Player Flash 5 or later.

[CONTENTS](#) 

XML.xmlDecl

Syntax `myXML.xmlDecl;`

Arguments None.

Description Property; sets and returns information about a document's XML declaration. After the XML document is parsed into an XML object, this property is set using the text of the document's XML declaration. This property is set using a string representation of the XML declaration, not an XML node object. If no XML declaration was encountered during a parse operation, the property is set to undefined. `XML.toString` outputs the contents of `XML.xmlDecl` before any other text in the XML object. If `XML.xmlDecl` contains the `undefined` type, no XML declaration is output.

Player Flash 5 or later.

Example The following example uses `XML.xmlDecl` to set the XML document declaration for an XML object:

```
myXML.xmlDecl = "<?xml version=\"1.0\" ?>";
```

See also

[XML.toString](#) [XML.docTypeDecl](#)

[CONTENTS](#) 

add

Syntax

```
string1 add string2
```

Arguments string1,2 Any string.

Description Operator; concatenates two or more strings. The add operator replaces the Flash 4 & operator; Flash 4 files using the & operator are automatically converted to use the add operator for string concatenation when brought into the Flash 5 authoring environment. However, the add operator is deprecated in Flash 5, and use of the + operator is recommended when creating content for the Flash 5 Player. Use the add operator to concatentate strings if you are creating content for Flash 4 or earlier versions of the Player.

Player Flash 4 or later.

See also

[+ \(addition\)](#)

CONTENTS 

__alpha

Syntax

```
instancename.__alpha  
instancename.__alpha = value;
```

Arguments `instancename` The name of a movie clip instance.

`value` A number from 0 to 100 specifying the alpha transparency.

Description Property; sets or retrieves the alpha transparency (`value`) of the movie clip. Valid values are 0 (fully transparent) to 100 (fully opaque). Objects in a movie clip with `__alpha` set to 0 are active, even though they are invisible. For example, a button in a movie clip with `__alpha` property set to 0 can still be clicked.

Player Flash 4 or later.

Example The following statements set the `__alpha` property of a movie clip named `star` to 30% when the button is clicked:

```
on(release) {  
    setProperty(star.__alpha = 30);  
}
```

or

```
on(release) {  
    star.__alpha = 30;  
}
```

[CONTENTS](#) 

__currentframe

Syntax

instancename.__currentframe

Arguments `instancename` The name of a movie clip instance.

Description Property (read-only); returns the number of the frame where the playhead is currently located in the Timeline.

Player Flash 4 or later.

Example The following example uses __currentframe to direct a movie to go five frames ahead of the frame containing the action:

```
gotoAndStop(__currentframe + 5);
```

[CONTENTS](#) 

__droptarget

Syntax

`draggableInstanceName.__droptarget`

Arguments `draggableInstanceName` The name of a movie clip instance that was the target of a `startDrag` action.

Description Property (read-only); returns the absolute path in slash syntax notation of the movie clip instance on which the `draggableInstanceName` was dropped. The `__droptarget` property always returns a path that starts with `/`. To compare the `__droptarget` property of an instance to a reference, use `eval` to convert the returned value from slash syntax to a reference.

Player Flash 4 or later.

Example The following example evaluates the `__droptarget` property of the `garbage` movie clip instance and uses `eval` to convert it from slash syntax to a dot syntax reference. The `garbage` reference is then compared to the reference to the `trash` movie clip instance. If the two references are equivalent, the visibility of `garbage` is set to `false`. If they are not equivalent, the `garbage` instance is reset to its original position.

```
if (eval(garbage.__droptarget) == _root.trash) { garbage._visible = false; } else { garbage._x = x_pos; garbage._y = y_pos; }
```

The variables `x_pos` and `y_pos` are set on frame 1 of the movie with the following script:

```
x_pos = garbage._x; y_pos = garbage._y;
```

See also

[startDrag](#)

[CONTENTS](#) 

eq (equal—string specific)

Syntax

```
expression1 eq expression2
```

Arguments expression1, expression2 Numbers, strings, or variables.

Description Comparison operator; compares two expressions for equality and returns true if expression1 is equal to expression2; otherwise, returns false.

Player Flash 1 or later. This operator has been deprecated in Flash 5; use of the new == (equality) operator is recommended.

See also

[== \(equality\)](#)

[CONTENTS](#) 

__focusrect

Syntax

```
__focusrect = Boolean;
```

Arguments Boolean true OR false.

Description Property (global); specifies whether a yellow rectangle appears around the button that has the current focus. The default value true (nonzero) displays a yellow rectangle around the currently focused button or text field as the user presses the Tab key to navigate. Specify false to display only the button "over" state (if any is defined) as users navigate.

Player Flash 4 or later.

[CONTENTS](#) 

_framesloaded

Syntax

instancename._framesloaded

Arguments `instancename` The name of the movie clip instance to be evaluated.

Description Property (read-only); the number of frames that have been loaded from a streaming movie. This property is useful for determining whether the contents of a specific frame, and all the frames before it, have loaded and are available locally in a user's browser. This property is useful for monitoring the download process of large movies. For example, you might want to display a message to users indicating that the movie is loading until a specified frame in the movie has finished loading.

Player Flash 4 or later.

Example The following is an example of using the _framesloaded property to coordinate the start of the movie to the number of frames loaded:

```
if (_framesloaded >= _totalframes) { gotoAndPlay ("Scene 1",  
"start"); } else { setProperty ("_root.loader", _xscale,  
(_framesloaded/_totalframes)*100); }
```

[CONTENTS](#) 

ge (greater than or equal to—string specific)

Syntax

`expression1 ge expression2`

Arguments expression1, expression2 Numbers, strings, or variables.

Description Operator (comparison); compares expression1 to expression2 and returns true if expression1 is greater than or equal to expression2; otherwise, returns false.

Player Flash 4 or later. This operator has been deprecated in Flash 5; use of the new >= operator is recommended.

See also

[>= \(greater than or equal to\)](#)

[CONTENTS](#) 

gt (greater than —string specific)

Syntax

expression1 gt expression2

Arguments expression1, expression2 Numbers, strings, or variables.

Description Operator (comparison); compares expression1 to expression2 and returns true if expression1 is greater than expression2; otherwise, returns false.

Player Flash 4 or later. This operator has been deprecated in Flash 5; use of the new > operator is recommended.

See also

[> \(greater than\)](#)

[CONTENTS](#) 

__height

Syntax

```
instancename.__height  
instancename.__height = value;
```

Arguments `instancename` An instance name of a movie clip for which the `__height` property is to be set or retrieved.

`value` An integer specifying the height of the movie in pixels.

Description Property; sets and retrieves the height of the space occupied by a movie's content. In previous versions of Flash, `__height` and `__width` were read-only properties; in Flash 5 these properties can be set.

Player Flash 4 or later.

Example The following code example sets the height and width of a movie clip when the user clicks the mouse:

```
onClipEvent(mouseDown) { __width=200; __height=200; }
```

[CONTENTS](#) 

`_highquality`

Syntax `_highquality = value;`

Arguments `value` The level of anti-aliasing applied to the movie. Specify 2 (BEST) to apply high quality with bitmap smoothing always on. Specify 1 (high quality) to apply anti-aliasing; this will smooth bitmaps if the movie does not contain animation. Specify 0 (low quality) to prevent anti-aliasing.

Description Property (global); specifies the level of anti-aliasing applied to the current movie.

Player Flash 4 or later.

See also

[_quality](#)

[toggleHighQuality](#)

[CONTENTS](#) 

Infinity

Syntax

Infinity

Arguments None.

Description Top-level variable; a predefined variable with the ECMA-262 value for infinity.

Player Flash 5 or later.

[CONTENTS](#) 

le (less than or equal to — string specific)

Syntax

`expression1 le expression2`

Arguments `expression1,expression2` Numbers, strings, or variables.

Description Operator (comparison); compares `expression1` to `expression2` and returns `true` if `expression1` is less than or equal to `expression2`; otherwise, returns `false`.

Player Flash 4 or later. This operator has been deprecated in Flash 5; use of the new `<=` operator is recommended.

See also

[<= \(less than or equal to\)](#)

[CONTENTS](#) 

_name

Syntax

`instancename._name`

`instancename._name = value;`

Arguments `instancename` An instance name of a movie clip for which the `_name` property is to be set or retrieved.

`value` A string that specifies a new instance name.

Description Property; specifies the movie clip instance name.

Player Flash 4 or later.

[CONTENTS](#) 

ne (not equal — string specific)

Syntax

`expression1 ne expression2`

Arguments expression1, expression2 Numbers, strings, or variables.

Description Operator (comparison); compares expression1 to expression2 and returns true if expression1 is not equal to expression2; otherwise, returns false.

Player Flash 4 or later. This operator has been deprecated in Flash 5; use of the new != (not equal) operator is recommended.

See also

[!= \(inequality\)](#)

[CONTENTS](#) 

new

Syntax

```
new constructor( ) ;
```

Arguments `constructor` A function followed by any optional arguments in the parentheses. The function is usually the name of the type of object (For example, Array, Math, Number, Object) to be constructed.

Description Operator; creates a new, initially anonymous object, calls the function identified by the `constructor` argument, passes any optional arguments in the parentheses, and passes the newly created object as a value of the keyword `this`. The constructor function can then use `this` to instantiate the new object.

The `_prototype_` property of the constructor function's object is copied into the `_proto_` property of the new object. As a result, the new object supports all of the methods and properties specified in the constructor function's Prototype object.

Player Flash 5 or later.

Example The following example creates the objects `book1` and `book2` using the `new` operator.

```
function Book(name, price)
{
    this.name = name;
    this.price = price;
}
book1 = new Book("Confederacy of Dunces", 19.95);
book2 = new Book("The Floating Opera", 10.95);
```

See also

[\[\] \(array access operator\)](#)

[{} \(object initializer\)](#)

The constructor method section within an object entry.

[CONTENTS](#) 

not

Syntax

not expression

Arguments expression Any variable or other expression that converts to a Boolean value.

Description Operator; performs a logical NOT operation in the Flash 4 Player.

Player Flash 4 or later. This operator has been deprecated in Flash 5; use of the new ! (logical NOT) operator is recommended.

See also

[! \(logical NOT\)](#)

[CONTENTS](#) 

null

Syntax

null

Arguments None.

Description Keyword; a special value that can be assigned to variables, or returned by a function if no data was provided. You can use null to represent values that are missing or do not have a defined data type.

Player Flash 5 or later.

Example In a numeric context, null evaluates to 0. Equality tests can be performed with null. In this statement, a binary tree node has no left child, so the field for its left child could be set to null.

```
if (tree.left == null) {  
    tree.left = new TreeNode();  
}
```

[CONTENTS](#) 

or

Syntax `condition1 or condition2`

Arguments `condition1,2` An expression that evaluates to `true` or `false`.

Description Operator; evaluates `condition1` and `condition2`, and if either expression is `true`, then the whole expression is `true`.

Player Flash 4 or later. This operator has been deprecated in Flash 5, and users are encouraged to make use of the new `||` operator.

See also

[!\(logical NOT\)](#)

[CONTENTS](#) 

__parent

Syntax

```
__parent.property = x  
__parent.__parent.property = x
```

Arguments `property` The property being specified for the current and parent movie clip.

`x` The value set for the property. This is an optional argument and may not need to be set, depending on the property.

Description Property; specifies or returns a reference to the movie clip that contains the current movie clip. The current movie clip is the movie clip containing the currently executing script. Use `__parent` to specify a relative path.

Player Flash 4 or later.

Example In the following example the movie clip `desk` is a child of the movie clip `classroom`. When the script below executes inside the movie clip `desk`, the playhead will jump to frame 10 in the Timeline of the movie clip `classroom`.

```
__parent.gotoAndStop(10);
```

See also

[_root](#)
[targetPath](#)

CONTENTS 

__quality

Syntax

```
__quality  
__quality = x;
```

Arguments x A string specifying one of the following values:

LOW Low rendering quality. Graphics are not antialiased, bitmaps are not smoothed.

MEDIUM Medium rendering quality. Graphics are antialiased using a 2x2 grid, but bitmaps are not smoothed. Suitable for movies that do not contain text.

HIGH High rendering quality. Graphics are antialiased using a 4x4 grid, and bitmaps are smoothed if the movie is static. This is the default rendering quality setting used by Flash.

BEST Very high rendering quality. Graphics are antialiased using a 4x4 grid, and bitmaps are always smoothed.

Description Property (global); sets or retrieves the rendering quality used for a movie.

Player Flash 5 or later.

Example The following example sets the rendering for oldQuality to HIGH:

```
oldQuality = __quality  
__quality = "HIGH";
```

See also

[_highquality](#)

[CONTENTS](#) 

__root

Syntax

```
__root;  
__root.movieClip;  
__root.action;
```

Arguments movieClip The instance name of a movie clip.

action The value set for the property. This is an optional argument and may not need to be set depending on the property.

Description Property; specifies or returns a reference to the root movie Timeline. If a movie has multiple levels, the root movie Timeline is on the level containing the currently executing script. For example, if a script in level 1 evaluates __root, level 1 is returned.

Specifying __root is the same as using the slash notation (/) to specify an absolute path within the current level.

Player Flash 4 or later.

Example The following example stops the Timeline of the level containing the currently executing script:

```
__root1.stop();
```

The following example sends the Timeline in the current level to frame 3:

```
__root.gotoAndStop(3);
```

See also

[_parent](#)
[targetPath](#)

CONTENTS 

__rotation

Syntax

```
instancename.__rotation  
instancename.__rotation = integer
```

Arguments integer The number of degrees to rotate the movie clip.

instancename The movie clip to rotate.

Description Property; specifies the rotation of the movie clip in degrees.

Player Flash 4 or later.

[CONTENTS](#) 

__soundbuftime

Syntax `__soundbuftime = integer;`

Arguments `integer` The number of seconds before the movie starts to stream.

Description Property (global); establishes the number of seconds of streaming sound to prebuffer. The default value is 5 seconds.

Player Flash 4 or later.

[CONTENTS](#) 

__target

Syntax

instancename.__target

Arguments instancename The name of a movie clip instance.

Description Property (read-only); returns the target path of the movie clip instance specified in the instancename argument.

Player Flash 4 or later.

[CONTENTS](#) 

this

Syntax

this

Arguments None.

Description Keyword; references an object or movie clip instance. The keyword this has the same purpose and function in ActionScript as it does in JavaScript, with some additional functionality. In ActionScript, when a script executes, this references the movie clip instance that contains the script. When used with a method invocation, this contains a reference to the object that contains the executed method.

Player Flash 5 or later.

Example In the following example, the keyword this references the Circle object:

```
function Circle(radius){ this.radius = radius; this.area = math.PI *  
radius * radius; }
```

In the following statement assigned to a frame, the keyword this references the current movie clip:

```
//sets the alpha property of the current movie clip to 20.  
this._alpha = 20;
```

In the following statement inside an onClipEvent handler, the keyword this references the current movie clip:

```
//when the movie clip loads, a startDrag operation is initiated  
for the current movie clip.onClipEvent (load) { startDrag (this,  
true); }
```

See also

[new](#)

[CONTENTS](#) 

__totalframes

Syntax

`instancename.__totalframes`

Arguments `instancename` The name of the movie clip to evaluate.

Description Property (read-only); evaluates the movie clip specified in the `instancename` argument and returns the total number of frames in the movie.

Player Flash 4 or later.

CONTENTS 

typeof

Syntax

```
typeof(expression);
```

Arguments `expression` A string, movie clip, object, or function.

Description Operator; a unary operator placed before a single argument. Causes Flash to evaluate `expression`; the result is a string specifying whether the expression is a string, movie clip, object, or function.

Player Flash 5 or later.

[CONTENTS](#) 

_url

Syntax `instancename._url`

Arguments `instancename` The target movie clip.

Description Property (read only); retrieves the URL of the SWF file from which the movie clip was downloaded.

Player Flash 4 or later?

[CONTENTS](#) 

__visible

Syntax

```
instancename._visible  
instancename._visible = Boolean;
```

Arguments Boolean Enter a true or false value to specify whether the movie clip is visible.

Description Property; determines whether or not the movie specified by the `instancename` argument is visible. Movie clips that are not visible (property set to false) are disabled. For example, a button in a movie clip with the `__visible` property set to false cannot be clicked.

Player Flash 4 or later.

[CONTENTS](#) 

void

Syntax

```
void (expression);
```

Arguments `expression` An expression of any value.

Description Operator; a unary operator that discards the `expression` value and returns an undefined value. The void operator is often used to evaluate a URL in order to test for side effects without displaying the evaluated expression in the browser window. The void operator is also used in comparisons using the `==` operator to test for undefined values.

Player Flash 5 or later.

[CONTENTS](#) 

__width

Syntax

```
instancename.__width  
instancename.__width =value;
```

Arguments value The width of the movie in pixels.

instancename An instance name of a movie clip for which the __width property is to be set or retrieved.

Description Property; sets the width of the movie. In previous versions of Flash, __height and __width were read-only properties; in Flash 5 they can be set as well as retrieved.

Player Flash 4 as a read-only property. In Flash 5 or later, this property can be set as well as retrieved.

Example The following code example sets the height and width properties of a movie clip when the user clicks the mouse:

```
onClipEvent(mouseDown) { __width=200; __height=200; }
```

See also

[__height](#)

[CONTENTS](#) 

__x

Syntax

```
instancename.__x  
instancename.__x = integer
```

Arguments *integer* The local x coordinate of the movie.

instancename The name of a movie clip instance.

Description Property; sets the x coordinate of movie relative to the local coordinates of the parent movie clip. If a movie clip is in the main Timeline, then its coordinate system refers to the upper left corner of the Stage as (0, 0). If the movie clip is inside another movie clip that has transformations, the movie clip is in the local coordinate system of the enclosing movie clip. Thus, for a movie clip rotated 90° counterclockwise, the movie clip's children inherit a coordinate system that is rotated 90° counterclockwise. The movie clip's coordinates refer to the registration point position.

Player Flash 3 or later.

See also

[_y](#)

[_xscale](#)

[CONTENTS](#) 

__xmouse

Syntax

instancename.__xmouse

Arguments instancename The name of a movie clip instance.

Description Property (read-only); returns the x coordinate of the mouse position.

Player Flash 5 or later.

See also

[Mouse \(object\)](#)

[__ymouse](#)

CONTENTS 

__xscale

Syntax

```
instancename.__xscale  
instancename.__xscale = percentage;
```

Arguments **percentage** A percentage value specifying the percentage for horizontally scaling the movie. The default value is 100.

instancename The name of a movie clip instance.

Description Property; determines the horizontal scale (**percentage**) of the movie clip as applied from the registration point of the movie clip. The default registration point is (0,0).

Scaling the local coordinate system affects the `_x` and `_y` property settings, which are defined in whole pixels. For example, if the parent movie clip is scaled to 50%, setting the `_x` property moves an object in the movie clip by half the number of pixels as it would if the movie were at 100%.

Player Flash 4 or later.

See also

[_xscale](#)

CONTENTS 

_y

Syntax

```
instancename._y  
instancename._y = integer;
```

Arguments *integer* The local y coordinate of the movie clip.

instancename The name of a movie clip instance.

Description Property; sets the y coordinate of movie relative to the local coordinates of the parent movie clip. If a movie clip is in the main Timeline, then its coordinate system refers to the upper left corner of the Stage as (0, 0). If the movie clip is inside another movie clip that has transformations, the movie clip is in the local coordinate system of the enclosing movie clip. Thus, for a movie clip rotated 90° counterclockwise, the movie clip's children inherit a coordinate system that is rotated 90° counterclockwise. The movie clip's coordinates refer to the registration point position.

Player Flash 3 or later.

See also

[_yscale](#)

[CONTENTS](#) 

__ymouse

Syntax

`instancename.__ymouse`

Arguments `instancename` The name of a movie clip instance.

Description Property (read-only); indicates the y coordinate of the mouse position.

Player Flash 5 or later.

See also

[Mouse \(object\)](#)

[__xmouse](#)

[CONTENTS](#) 

__yscale

Syntax

```
instancename.__yscale  
instancename.__yscale = percentage ;
```

Arguments **percentage** A percentage value specifying the percentage for vertically scaling the movie. The default value is 100.

instancename The name of a movie clip instance.

Description Property; sets the vertical scale (**percentage**) of the movie clip as applied from the registration point of the movie clip. The default registration point is (0,0).

Scaling the local coordinate system affects the `_x` and `_y` property settings, which are defined in whole pixels. For example, if the parent movie clip is scaled to 50%, setting the `_x` property moves an object in the movie clip by half the number of pixels as it would if the movie were at 100%.

Player Flash 4 or later.

See also

[_x](#)

[_y](#)

[CONTENTS](#) 

Object (object)

The generic Object object is at the root of the ActionScript class hierarchy. The functionality of the generic Object object is a small subset of that provided by the JavaScript Object object.

The generic Object object requires the Flash 5 Player.

Method summary for the Object object

toString	Converts the specified object to a string, and returns it.
valueOf	Returns the primitive value of an Object object.

Constructor for the Object object

Syntax

```
new Object();  
new Object(value);
```

Arguments `value` A number, Boolean, or string to be converted to an object. This argument is optional. If you do not specify `value`, the constructor creates a new object with no defined properties.

Description Constructor; creates a new Object object.

Player Flash 5 or later.

See also

[Sound.setTransform](#) [Color.setTransform](#)

CONTENTS 

and

Syntax `condition1 and condition2`

Arguments `condition1, condition2` Conditions or expressions that evaluate to `true` or `false`.

Description Operator; performs a logical AND operation in the Flash 4 Player. If both expressions evaluate to `true`, then the entire expression is `true`.

Player Flash 4 or later. This operator has been deprecated in Flash 5, and users are encouraged to make use of the new `&&` operator.

See also

[&& \(short-circuit AND\)](#)

[CONTENTS](#) 

MovieClip.loadMovie

Syntax

```
anyMovieClip.loadMovie(url [,variables]);
```

Arguments `url` An absolute or relative URL for the SWF file to load. A relative path must be relative to the SWF. The URL must be in the same subdomain as the URL where the movie currently resides. For use in the Flash Player or for testing in test-movie mode in the Flash authoring environment, all SWF files must be stored in the same folder, and the file names cannot include folder or disk drive specifications.

`variables` An optional argument specifying a method for sending variables associated with the movie to load. The argument must be the string "GET" or "POST." If there are no variables, omit this argument; otherwise, specify whether to load variables using a GET or POST method. GET appends the variables to the end of the URL and is used for small numbers of variables. POST sends the variables in a separate HTTP header and is used for long strings of variables.

Description Method; plays additional movies without closing the Flash Player. Normally, the Flash Player displays a single Flash Player movie (SWF file) and then closes. The loadMovie method allows you display several movies at once or switch between movies without loading another HTML document.

Use the unloadMovie action to remove movies loaded with the loadMovie action.

Use the loadVariables method to keep the active movie, and update the variables with new values.

Player Flash 5 or later.

See also

[MovieClip.loadVariables](#) [MovieClip.unloadMovie](#)

CONTENTS 

break

Syntax

`break;`

Arguments

None.

Description Action; appears within a loop (for, for..in, do...while or while). The break action instructs Flash to skip the rest of the loop body, stop the looping action, and execute the statement following the loop statement. Use the break action to break out of a series of nested loops.

Player Flash 4 or later.

Example The following example uses the break action to exit an otherwise infinite loop:

```
i = 0; while (true) { if (i >= 100) { break; } i++; }
```

[CONTENTS](#) 

call

Syntax

```
call(frame);
```

Arguments `frame` The name or number of the frame to call into the context of the script.

Description Action; switches the context from the current script to the script attached to the frame being called. Local variables will not exist once the script is finished executing.

Player Flash 4 or later. This action is deprecated in Flash 5, and it is recommended that you use the `function` action.

See also

[function](#)

[CONTENTS](#) 

function

Syntax

```
function functionname ([argument0, argument1,...argumentN]){  
    statement(s)  
}  
function ([argument0, argument1,...argumentN]){  
    statement(s)  
}
```

Arguments `functionname` The name of the new function.

`argument` Zero or more strings, numbers, or objects to pass the function.

`statements` Zero or more ActionScript statements you have defined for the body of the function.

Description Action; a set of statements that you define to perform a certain task. You can declare, or define, a function in one location and call, or invoke, it from different scripts in a movie. When you define a function, you can also specify arguments for the function. Arguments are placeholders for values on which the function will operate. You can pass a function different arguments, also called parameters, each time you call it.

Use the return action in a functions `statement(s)` to cause a function to return, or generate, a value.

Usage 1: Declares a function with the specified `functionname`, `arguments`, and `statement(s)`. When a function is called, the function declaration is invoked. Forward referencing is permitted; within the same Action list, a function may be declared after it is called. A function declaration replaces any prior declaration of the same function. You can use this syntax wherever a statement is permitted.

Usage 2: Creates an anonymous function and returns it. This syntax is used in expressions, and is particularly useful for installing methods in objects.

Player Flash 5 or later.

Example (Usage 1) The following example defines the function `sqr`, which accepts one argument, and returns the square($x*x$) of the argument. Note that if the function is declared and used in the same script, the function declaration may appear after using the function.

```
y=sqr(3);  
function sqr(x) {  
    return x*x;  
}
```

(Usage 2) The following function defines a Circle object:

```
function Circle(radius) {  
    this.radius = radius;  
}
```

The following statement defines an anonymous function that calculates the area of a circle and attaches it to the object Circle as a method:

```
Circle.prototype.area = function () {return Math.PI * this.radius * this.radius}
```

[CONTENTS](#) 

" " (string delimiter)

Syntax

"text"

Arguments text Any text.

Description String delimiter; when used before and after a string, quotes indicate that the string is a literal—not a variable, numerical value, or other ActionScript element.

Player Flash 4 or later.

Example This statement uses quotes to indicate that the string "Prince Edward Island" is a literal string, and not the value of a variable:

```
province = "Prince Edward Island"
```

See also

[String \(object\)](#)

[String \(function\)](#)

[CONTENTS](#) 

`_level`

Syntax

`_levelN;`

Arguments N A nonnegative integer specifying a depth level. By default, `_level` is set to 0, the movie at the base of the hierarchy.

Description Property; a reference to the root movie Timeline of `levelN`. You must load movies using the `loadMovie` action, before targeting them using the `_level` property.

In the Flash Player, movies are assigned a number according to the order in which they were loaded. The movie that was loaded first is loaded at the bottom level, level 0. The movie in level 0 sets the frame rate, background color, and frame size for all subsequently loaded movies. Movies are then stacked in higher numbered levels above the movie in level 0. The level where a movie clip resides is also referred to as the depth level or depth.

Player Flash 4 or later.

Example The following example stops the Timeline of the movie in level 0:

```
_level0.stop();
```

The following example sends the Timeline of the movie in level 4 to frame 5. The movie in level 4 must have previously been loaded with a `loadMovie` action:

```
_level4.gotoAndStop(5);
```

See also

[loadMovie](#) [MovieClip.swapDepths](#)

CONTENTS 

MovieClip.swapDepths

Syntax

```
anyMovieClip.swapDepths(depth);  
anyMovieClip.swapDepths(target);
```

Arguments **target** The movie clip instance whose depth that is being swapped by the instance specified in anyMovieClip. Both instances must have the same parent movie clip.

depth A number specifying the depth level where the anyMovieClip is to be placed.

Description Method; swaps the stacking, or z, order (depth level) of the specified instance with the movie specified by the **target** argument, or with the movie that currently occupies the **depth** level specified in the argument. Both movies must have the same parent movie clip. Swapping the depth level of movie clips has the effect of moving one movie in front of or behind the other. If a movie is tweening when this method is called, the tweening is stopped.

Player Flash 5 or later.

See also

[_level](#)

CONTENTS 

Array.unshift

Syntax

```
myArray.unshift(value1,value2,...valueN);
```

Arguments `value1,...,valueN` One or more numbers, elements, or variables to be inserted at the beginning of the array.

Description Method; adds one or more elements to the beginning of an array and returns the array's new length.

Player Flash 5 or later.

[CONTENTS](#) 

stop

Syntax

```
stop;
```

Arguments None.

Description Action; stops the movie that is currently playing. The most common use of this action is to control movie clips with buttons.

Player Flash 3 or later.

[CONTENTS](#) 

Sample entry for most ActionScript elements

The following sample dictionary entry explains the conventions used for all ActionScript elements that are not objects.

Entry title

All entries are listed alphabetically. The alphabetization ignores capitalization, leading underscores, and so on.

Syntax The "Syntax" section provides correct syntax for using the ActionScript element in your code. The code portion of the syntax is in `code font`, and the arguments you must provide are in *italicized code font*. Brackets indicate optional arguments.

Arguments This section describes any arguments listed in the syntax.

Description This section identifies the element (for example, as an operator, method, function, or other element) and then describes how the element is used.

Player This section tells which versions of the Player support the element. This is not the same as the version of Flash used to author content. For example, if you are creating content for the Flash 4 Player using the Flash 5 authoring tool, you cannot use ActionScript elements that are only available to the Flash 5 Player.

With the introduction of Flash 5 ActionScript, some Flash 4 (and earlier) ActionScript elements have been deprecated. Although deprecated elements are still supported by the Flash 5 Player, it is recommended that you use the new Flash 5 elements.

In addition, operator functionality has been greatly expanded in Flash 5. Not only have many new mathematical operators been introduced, but some of the older operators are now capable of handling additional data types. To maintain data type consistency, Flash 4 files are automatically modified when imported into the Flash 5 authoring environment, but these modifications will not affect the functionality of the original script. For more information, see the entries for + (addition), < (less than), > (greater than), <= (less than or equal to), >= (greater than or equal to), != (inequality), and = (equality).

Example This section provides a code sample demonstrating how to use the element.

See also This section lists related ActionScript dictionary entries.

[CONTENTS](#)  [NEXT](#) 

Sample entry for objects

The following sample dictionary entry explains the conventions used for predefined ActionScript objects. Objects are listed alphabetically with all other elements in the dictionary.

Entry title

The entry title provides the name of the object. The object name is followed by a paragraph containing general information about the object.

Method and property summary tables

Each object entry contains a table listing all of the methods associated with the object. If the object has properties (often constants), these elements are summarized in an additional table. All of the methods and properties listed in these tables also have their own dictionary entries, which follow the object entry.

Constructor

If the object requires you to use a constructor to access its methods and properties, the constructor is described at the end of the object entry. This description has all of the standard elements (syntax description, and so on) of other dictionary entries.

Method and property listings

The methods and properties of an object are listed alphabetically after the object entry.

[CONTENTS](#) 

[PREVIOUS](#) 

[NEXT](#) 